VISIT…

LANZAROTE
Caliente.COM

Frank Dignum   Jeff Bradshaw
Barry Silverman   Willem van Doesburg (Eds.)

Agents for Games and Simulations

Trends in Techniques, Concepts and Design

Springer

Volume Editors

Frank Dignum
Utrecht University
Dept. of Information and Computing Sciences
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
E-mail: dignum@cs.uu.nl

Jeff Bradshaw
Florida Institute for Human & Machine Cognition
40 South Alcaniz Street, Pensacola, FL 32502 USA
E-mail: jbradshaw@ihmc.us

Barry Silverman
University of Pennsylvania/Systems, Electrical and Systems Engineering
Engineering Dept. Philadelphia
Towne Building, Room 251, 220 South 33rd Street
Philadelphia, PA 19104-6315, USA
E-mail: barryg@seas.upenn.edu

Willem van Doesburg
TNO Defence
Department of Training and Instruction
Security and Safety
P.O. Box 23, 3769 ZG Soesterberg, The Netherlands
E-mail: willem.vandoesburg@tno.nl

Library of Congress Control Number: 2009940795

CR Subject Classification (1998): I.2, I.2.11, I.2.6, I.2.1, F.1.1, I.6, H.2.8

LNCS Sublibrary: SL 7 – Artificial Intelligence

ISSN 0302-9743
ISBN-10 3-642-11197-1 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-11197-6 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12811878 06/3180 5 4 3 2 1 0

Preface

Multi-agent system research offers a promising technology for implementing cognitive intelligent non-playing characters. However, the technologies used in game engines and multi-agent platforms are not readily compatible due to some inherent differences in concerns. Where game engines focus on real-time aspects and thus propagate efficiency and central control, multi-agent platforms assume autonomy of the agents. Increased autonomy and intelligence may offer benefits for a more compelling gameplay and may even be necessary for serious games. However, it raises problems when current game design techniques are used to incorporate state-of-the-art multi-agent system technology. A very similar argument can be given for agent-based (social) simulations.

This volume contains the papers presented at AGS 2009: The First International Workshop on Agents for Games and Simulations held on May 11 in Budapest. In this workshop people came together to address the particular challenges of using agent technology for games and simulations. Submissions were invited for the following three main themes:

Technical: Connecting agent platforms to games and simulation engines; who is in control?, Are actions synchronous or asynchronous? How to monitor results of actions. Can agents communicate through the agent platform? How efficient should the agents be?

Conceptual: What information is available for the agents from the game or simulation engine? How to balance reaction to events of the game or simulation with goal-directed behavior. Ontological differences between agents and game/simulation information.

Design: How to design games/simulations containing intelligent agents. How to design agents that are embedded in other systems. Of course we also welcomed papers about experiences in the use of agents in games and simulations. Both successes as well as "failures" were welcome, as both can help us better understand what are the key issues in combining agents with game and simulation engines.

We received 17 submissions of high quality covering many of the aspects mentioned above. Each submission was reviewed by at least two Program Committee members. We accepted 11 papers for presentation, which can be found in these proceedings. We also invited some authors who covered aspects that were considered to be important in this area, but were not present in the workshop yet. Together the selection of papers in the present volume constitutes a good overview of the state of the art in this area. Among the papers we find a strong example of a description of middleware that is used to connect agents to Unreal Tournament (an extension of Gamebots). But we also have a paper describing how to design agents for games that have to behave according to cognitive and emotive theories in order to mimic or respond to human behavior in a naturalistic way. Several papers describe experiences with particular agent models used for games and simulations. Finally, the book also contains a paper that discusses how to evaluate the behavior of the agents in the games. When are they performing “well”? All in all we are very happy with the papers contained in this volume. We are sure they form a valuable starting point for people that want to combine agent technology with (serious) games. Finally, we would like to thank the Program Committee members, without whom the reviewing would not have been possible, and for their valuable comments on all papers allowing for a tough selection of the best paper.

October 2009 Frank Dignum

Conference Organization

Program Chairs

Frank Dignum
Jeff Bradshaw
Barry Silverman
Willem van Doesburg

Program Committee

Elisabeth Andre	DFKI, Germany
Ruth Aylett	Heriot-Watt University, UK
Andre Campos	UFRN, Brazil
Bill Clancey	NASA, USA
Rosaria Conte	ISTC-CNR, Italy
Vincent Corruble	LIP6, France
Yves Demazeau	CNRS-LIG, Grenoble
Virginia Dignum	Utrecht University, The Netherlands
Alexis Drogoul	LIP6, France
Bruce Edmonds	MMU, UK
Corinna Elsenbroich	University of Surrey, UK
Klaus Fischer	DFKI, Germany
Hiromitsu Hattori	Kyoto University, Japan
Koen Hindriks	Delft University, The Netherlands
Wander Jager	Groningen University, The Netherlands
Stefan Kopp	University of Bielefeld, Germany
Michael Lewis	University of Pittsburg, USA
Scott Moss	MMU, UK
Emma Norling	MMU, UK
Anton Nijholt	UT, The Netherlands
Ana Paiva	IST, Portugal
Michal Pechoucek	CTU, Czech Republic
David Pynadath	USC, USA
Geber Ramalho	Brazil
Gopal Ramchurn	University of Southampton, UK
Avi Rosenfeld	JCT, Israel
David Sarne	Bar Ilan University, Israel
Pjotr van Schothorst	VSTEP, The Netherlands
Maarten Sierhuis	NASA, USA
Pieter Spronck	Tilburg University, The Netherlands

Katia Sycara	CMU, USA
Duane Szafron	University of Alberta, Canada
Max Tsvetovat	George Mason University, USA

External Reviewers

Leon-Soto, Esteban
Marsella, Stacy
Silverman, Barry
Westra, Joost

Table of Contents

Pogamut 3 Can Assist Developers in Building AI (Not Only) for Their Videogame Agents

Jakub Gemrot, Rudolf Kadlec, Michal Bída, Ondřej Burkert, Radek Píbil, Jan Havlíček, Lukáš Zemčák, Juraj Šimlovič, Radim Vansa, Michal Štolba, Tomáš Plch, and Cyril Brom

Charles University in Prague, Faculty of Mathematics and Physics
Malostranské nám. 2/25, Prague, Czech Republic

Abstract. Many research projects oriented on control mechanisms of virtual agents in videogames have emerged in recent years. However, this boost has not been accompanied with the emergence of toolkits supporting development of these projects, slowing down the progress in the field. Here, we present Pogamut 3, an open source platform for rapid development of behaviour for virtual agents embodied in a 3D environment of the Unreal Tournament 2004 videogame. Pogamut 3 is designed to support research as well as educational projects. The paper also briefly touches extensions of Pogamut 3; the ACT-R integration, the emotional model ALMA integration, support for control of avatars at the level of gestures, and a toolkit for developing educational scenarios concerning orientation in urban areas. These extensions make Pogamut 3 applicable beyond the domain of computer games.

Keywords: Virtual agents, behaviour, control mechanisms, Unreal Tournament, 3D environment, agent development toolkit, emotions, ACT-R, gestures, education, ALMA.

1 Introduction

The development of control mechanisms for videogame agents (that is, their “artificial intelligence”) is hard by itself. The developer’s work is typically further complicated by the fact that many tedious technical issues, out of the main scope of their work, have to be solved. For instance, they have to integrate their agent within a particular 3D environment, develop at least simple debugging tools or write a code for low-level movement of their agents. This list may continue for pages and developers address these issues over and over, in most cases a waste of energy and time.

For last four years, we have been developing the Pogamut toolkit, which provides general solutions for many of these issues, allowing developers to focus on their main goals. A new major version, Pogamut 3, has been released. Importantly, Pogamut 3 is designed not only for advanced researchers, but also for newcomers. Pogamut 3 can help them to build their first virtual agents, a feature which makes it applicable as a toolkit for training in university and high-school courses, as demonstrated in [9].

The purpose of this paper is to review the main features of Pogamut 3, its architecture, extensions, and work in progress. Section 2 introduces the main features. Section 3 briefly discusses applications of Pogamut 2 and Pogamut 3. Section 4 introduces new extensions added to Pogamut 3 recently. Section 5 discusses related work. Section 6 discusses work in progress and concludes the paper.

This paper is an extension of a short paper presented in [21]. More information about Pogamut 3 can be found at the project homepage [3], where an installer is available for download. The project's page features tutorial videos, learning material about the platform and forums, which provide the always-needed support and help to build the community around Pogamut 3.

2 Features of Pogamut 3

The agent development cycle can be conceptualized into five stages: (1) inventing an agent, (2) implementing the agent, (3) debugging the implemented agent, (4) tuning the agent's parameters, and (5) validating the agent through series of experiments. Individual features of Pogamut 3 were purposely designed to provide the support during the last four stages.

The features of Pogamut 3 include:

1) a binding to the virtual world of the Unreal Tournament 2004 videogame (UT2004) [17],
2) an integrated development environment (IDE) with a support for debugging (Fig. 2),
3) a library with sensory-motor primitives, path-finding algorithms, and a support for shooting behaviour and weapon handling,
4) a connection to the POSH [11], which is a reactive planner for controlling behaviour of agents, and a visual editor for POSH plans (Fig. 4),
5) a support for running experiments, including distributed experiments running on a GRID.

Technically, the Pogamut 3 toolkit integrates five main components: *UT2004*, *GameBots2004*, the *GaviaLib library*, the *Pogamut agent*, and the *IDE* (Fig. 1). These components implement the features mentioned above, enabling a user to create, debug, and evaluate his or her agents conveniently.

In general, the components of the Pogamut 3 toolkit can be used *per partes*. For instance, one can use GameBots2004 independently as well as a part of the GaviaLib library or the IDE. We will return to this point later.

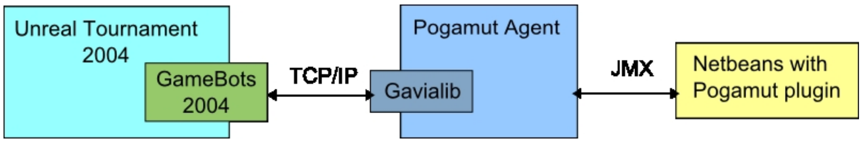
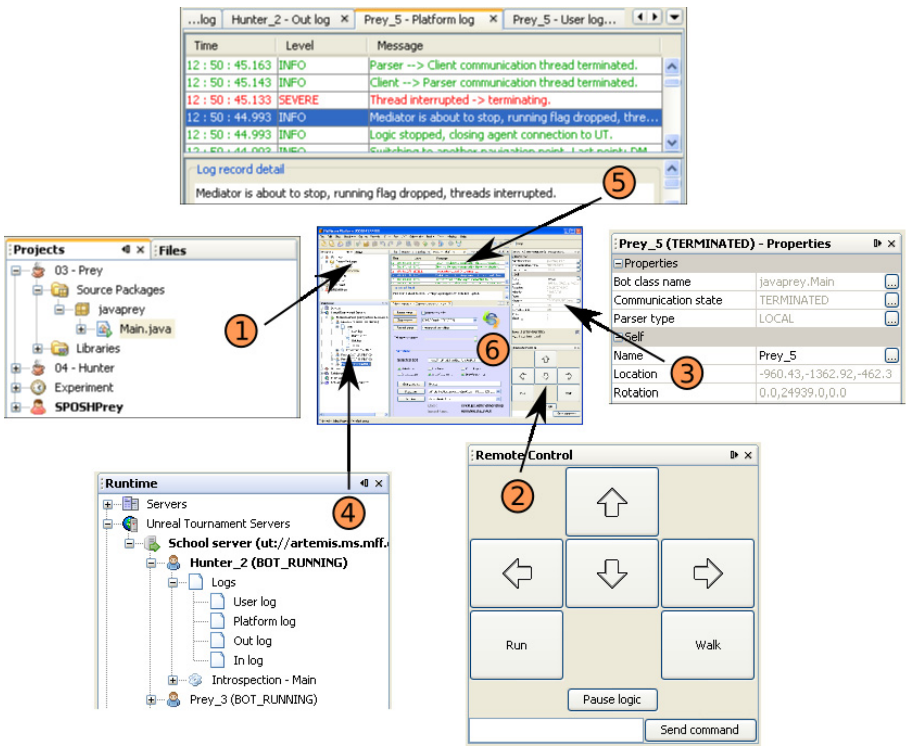


Fig. 1. Pogamut architecture



```
Time           Level    Message
12 : 50 : 45.163 INFO     Parser --> Client communication thread terminated.
12 : 50 : 45.143 INFO     Client --> Parser communication thread terminated.
12 : 50 : 45.133 SEVERE   Thread interrupted -> terminating.
12 : 50 : 44.993 INFO     Mediator is about to stop, running flag dropped, thre...
12 : 50 : 44.993 INFO     Logic stopped, closing agent connection to UT.
```

Log record detail
Mediator is about to stop, running flag dropped, threads interrupted.

Prey_5 (TERMINATED) - Properties

Properties	
Bot class name	javaprey.Main
Communication state	TERMINATED
Parser type	LOCAL
Self	
Name	Prey_5
Location	-960.43,-1362.92,-462.3
Rotation	0.0,24939.0,0.0

Fig. 2. Pogamut 3 NetBeans plugin (6) and its parts (1-5). The parts are described in the text.

2.1 Virtual Environment of Pogamut 3

UT2004 is a well-known action videogame with so-called "partly opened" code. In particular, the game features UnrealScript, a native scripting language developed by the authors of UT2004, in which a substantial part of the game is programmed; almost everything except of the graphical and the physical engine. Users can modify the code in UnrealScript. Additionally, the game features a lot of pre-built objects, maps, and a level editor. These two points allow users to create new game extensions and blend them with the original game content. In the context of the Pogamut 3 toolkit, the game provides the virtual environment for running agents.

Actually, there are more games with partly or fully opened codes, for instance Halo 2 [12] or Quake 3 [25]. Each of these games has its advantages and disadvantages – there is no single winner. We chose UT2004 since its user community is large and it is relatively new offering better graphical experience comparing e.g. to Quake 3. Concerning UT2004's disadvantages, it should be mentioned, for instance, that the whole code of Quake 3's engine is available, which is not the case of UT2004. Halo2 would likely be a better platform than UT2004 for developing tactic AI.

2.2 Interface to the Virtual World

One can implement agents directly in UnrealScript; however, it is often advantageous, both for educational as well as experimental reasons, to develop the agents using a different, external mechanism. This means that one has to create a binding to the UT2004 game. Several years ago, Adobbati et al. developed a generic binding for this purpose called GameBots [1]. It was swiftly adopted by the UT community. The original GameBots worked with UT99. GameBots2004 is our customized extension of the original GameBots, which connects the rest of the Pogamut toolkit to the UT2004.

GameBots2004 exports information about the game environment through a TCP/IP text based protocol, allowing users to connect to UT employing the client-server architecture. This means that the user implements an agent's control mechanisms at the client and uses GameBots2004 as the server.

Importantly, GameBots2004 can be used separately without other components of Pogamut 3. Thus, one can connect his/her own development environment to UT2004. On the other hand, if someone aims at connecting Pogamut 3 to other virtual environment, he/she has to develop his/her own world-interface replacing GameBots2004.

2.3 GaviaLib Library

The GaviaLib library has been developed as a general purpose Java library for connecting agents to almost any virtual environment, i.e. it is a generic interface for accessing a wide range of possible worlds. Only mild assumptions have been imposed upon virtual environments; in a nutshell, an environment has to work with objects and be able to provide information in an event-based manner.

GaviaLib provides:

- Abstract agent implementation, which handles the agent's life cycle, allows for the remote control of the agent through JMX [40], a standard Java protocol for remote instrumentation of programs, and defines a common set of logs,
- Agent's interface to the world, which manages object and event identities and notifies the agent about important object events (e.g. an object has appeared/disappeared/been changed),
- XML/XSLT framework for the definition of the world's objects and events.

What deserves most attention is the agent's interface. This interface provides API for listening for events and for querying object instances. Both of these use cases utilize Java class hierarchy allowing for different levels of abstraction, an analogy to semantic ontologies. This feature can be illustrated on the following example. Let us assume a simulator of a virtual world that can be populated by items such as fruits and vehicles, but currently only apples and cars are supported. Concerning agents that are to populate this world, this means that designers must provide a way for the agents to recognize items of these two kinds. When GaviaLib is used, designers have to create following classes representing the objects' types and their categories: "item", "fruit", "vehicle", "apple", and "car". Utilizing Java class inheritance, designers will further

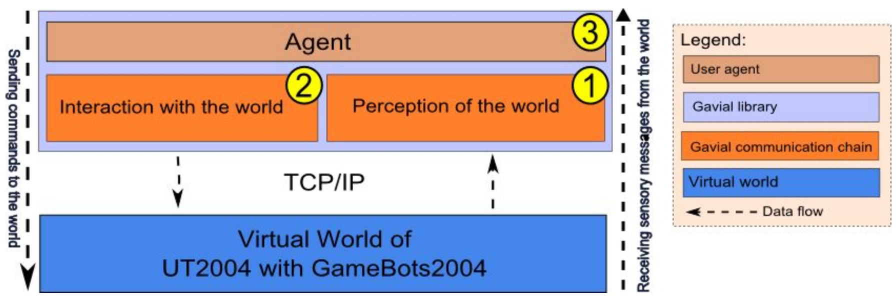

Agent
3
Interaction with the world
2
Perception of the world
1
TCP/IP
Virtual World of UT2004 with GameBots2004
Sending commands to the world
Receiving sensory messages from the world
Legend:
User agent
Gavial library
Gavial communication chain
Virtual world
Data flow

Fig. 3. High-level GaviaLib architecture overview. (1) Perceptual part servers to process information messages from the virtual world that Pogamut 3 is connected to. (2) Interaction part sends commands to the virtual world. (3) Agent abstractions comprise stubs of agents, defining high-level interface for agents' minds.

define that classes "vehicle" extends "item", "fruit" extends "item", "apple" extends "fruit", and "car" extends "vehicle". GaviaLib then propagates events according to this ontology; for instance, when an event happens on the object of class "apple", it is propagated also to "fruit" and "item". This makes the event model of GaviaLib flexible as the user may listen on all "fruit" events by attaching a listener on the class "fruit" instead of attaching the listener on every instance of fruit. This feature is not available when using GameBots2004 alone. This implementation also allows for custom extensions to existing code. When another designer creates a new class "banana" that extends "fruit", all existing listeners on "fruit" will automatically be able to report events that have happened on all bananas.

The GaviaLib's layered architecture (Fig. 3) makes it possible to use different means of communication with the virtual world. A GaviaLib agent can be connected through plain text TCP/IP protocols (like GameBots2004), CORBA or, in the case of Java environments, it can reside in the same Java Virtual Machine as the world simulator and communicate directly through method calls.

GaviaLib has also other features. These are fully detailed in [22].

The possibility of general usage of the library will be known only after rigorous testing with various virtual environments. Presently, the only stable binding featured by GaviaLib is the UT2004 binding, which employs Gamebots2004, as detailed in Sec. 2.4. However, we also have a work-in-progress binding with the Defcon game [27]. Defcon is simulation of a global thermonuclear conflict played in real-time. The player takes a role of a commander in charge of military forces under the flag of one nuclear nation. The successful binding of GaviaLib to Defcon would provide evidence that the library could be used as a general middleware for connecting to virtual environments. The investigation of GaviaLib possibilities on a rigorous basis presents future work.

2.4 UT2004 Agent

The UT2004 agent is a set of Java classes and interfaces derived from the basic classes of the GaviaLib library. The UT2004 agent is derived from all Parts (1) – (3) from Fig. 3, adding UT2004 specific features, such as UT2004's sensory-motor

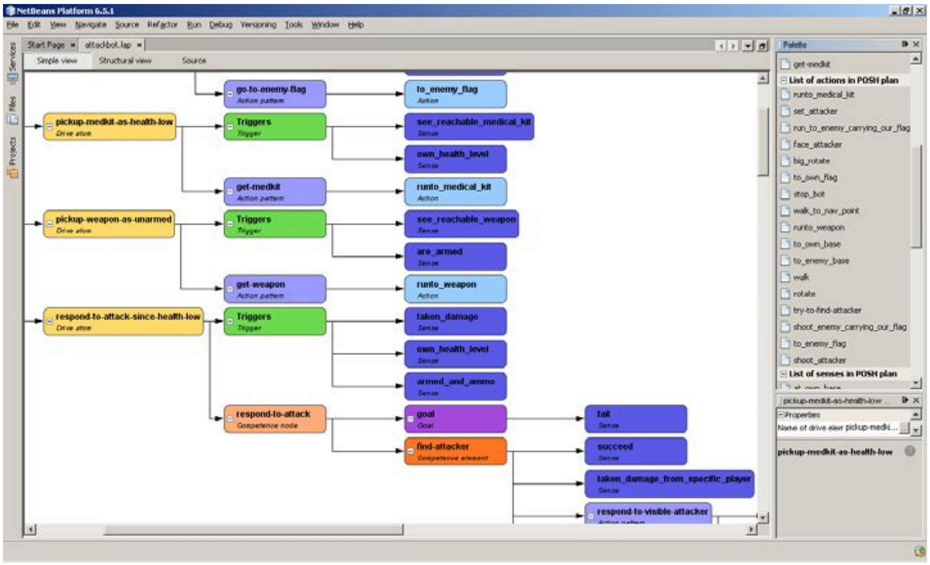


Fig. 4. Screenshot of POSH visual editor. A part of the plan of Hunter agent, which is a standard Pogamut example agent, is depicted.

primitives, the navigation utilizing the internal UT2004 way-point system and auxiliary classes providing information about the game rules. Thus, if a user wants to connect Pogamut to a different environment, he/she has to do both of these things: a) create a new interface to the virtual world (i.e. replace GameBots2004), b) create domain specific mechanisms at the side of the GaviaLib library (i.e. replace UT2004 agent).

Developers can program UT2004 agents using the classes of the Pogamut agent; they can implement an agent's control mechanism directly in Java or use the reactive planner POSH. POSH has been developed as an independent action selection planner by Joanna Bryson [11] and Pogamut 3 exploits it as a plug-in. From our perspective, the main advantage of this planner is that it is easy to understand for beginners allowing them to program their first agents quickly and easily comparing e.g. to Soar [44], Drools rules [29] or Jadex [4]. Similarly to some but not all of the other planners and reasoners, it enforces the design that clearly separates low-level actions and sensory primitives (coded in Java or Python) from a high level hierarchical plan of an agent's behaviour, simplifying the design process. POSH also lacks some features of more advanced planners and rule-based engines such as variable binding at the level of plans. This is often an advantage for beginners; some of these features may be unintelligible to them. However, at the same time, these features may be vital for advanced users. For those who already advance, it may be an option to use plain Java or another planner instead of POSH.

The editor for POSH plans is available (Fig. 4).

2.5 Integrated Development Environment

The IDE is developed as a NetBeans plugin [41] (Fig. 2). It can communicate with running agents via JMX. The IDE offers multiple features for a designer, such as empty agent project templates, example agents (e.g. hunter and prey agents or a khepera-like agent) (Fig. 2, Window 1), management of UT2004 servers, list of running agents (Fig. 2, Window 4), introspection of agent's variables and a quick access to general agent properties (Fig. 2, Window 3), the step-by-step debugging, log viewers (Fig. 2, Window 5), and the manual controller of agent's position (Fig. 2, Window 2). However, the Pogamut agents can also be developed without this plug-in in any other Java IDE using only GaviaLib with the UT2004 extension.

3 Applications and Scalability

Pogamut 2 (i.e. the previous version) has been used in many research projects [e.g. 42, 38], including student projects at our university ranging from utilizing evolutionary techniques for agent development [31], trough fuzzy rule control of behaviour of agents [39], up to episodic memory modelling [13]. Pogamut 2 has been employed for training in university and high-school courses [9, see also 7 for supplementary materials]. It was also picked as a base for 2K BotPrize competition[1], where participants were challenged to develop an agent that can fool the panel of judges that it is a human.

Some projects have been completed using Pogamut 3 (i.e. the present version), including a prototype of an educational game for teaching students the basics of developing virtual agents and virtual storytelling [8]. However, Pogamut 3 is presently more buggy than the previous version (although this may change soon).

The number of UT2004 agents a user can control via Pogamut 3 depends on the agents' complexity. Generally, Pogamut 3 cannot be expected to handle more than 10 agents at the same time. However, Pogamut 3, as well as Pogamut 2, allows for running both Pogamut agents as well as UT agents (i.e. agents programmed directly in the Unreal Script) simultaneously and the number of UT agents is limited only by the UT2004 server ability to handle the load.

The synchronous perception-action cycle for Pogamut 3 agents lasts 250 ms. This constant can be decreased to about 100 ms depending on the computer and the number and complexity of agents within the simulation. Decreasing the constant below this number can render the application unstable mainly due to GameBots2004 and UT2004 limitations. Asynchronous messages are processed instantaneously; the time lag caused by the propagation of the event via GameBots2004 and GaviaLib is in the order of milliseconds depending on the computer hardware used.

4 Pogamut's Extensions

Recently, several extensions to Pogamut 3 have been completed. The purpose of these extensions is to make Pogamut 3 useful also beyond the domain of gaming AI. The extensions have been developed independently: they are not integrated in a single

[1] http://botprize.org (27.9.2009)

package. Moreover, they are not included into the Pogamut 3 release version at the time of writing of this paper, but they may be available for download in coming months. In this section, we will review these extensions in further detail: ACT-R Integration, Gestures module, ALMA Integration, and Educational Scenarios.

4.1 ACT-R Integration

Recently, it has been proposed that computational cognitive science modelling can benefit from implementing models using virtual agents [10, 30]. This means, a neuro-/psychologically plausible model can be integrated within a virtual agent's "mind" and then tested trough scenarios in virtual worlds. Advantages of this approach have been discussed in [10]. One potential way how to do this is to adopt a general cognitive architecture, such as Soar [33, 44], ACT-R [2] or LIDA [19], as the main architecture of a virtual agent's mind. These architectures are somewhat plausible and tend to be modular; hence one can embed his or her model in one of the modules without loosing much of its original plausibility. In fact, Jilk and colleagues [30] did exactly this for a model of the visual cortex.

Pogamut 3 newly offers a binding to ACT-R cognitive architecture, so called PojACT-R [50] (Fig. 5). Because Pogamut is written in Java, we used jACT-R [24], the Java implementation of ACT-R. We run jACT-R in a separate thread. The important limitation of PojACT-R is that the cognitive cycle of ACT-R lasts 50 ms, while the Pogamut's cycle is longer (about 100-250 ms). This issue is partly ameliorated by a special type of perception buffers implemented by the default PojACT-R agent. These buffers hold perceptual information from the UT2004 temporarily and pass it to the jACT-R in a step-wise manner while giving high priority to asynchronous messages.

To demonstrate that PojACT-R works, we implemented a simple Hunter agent featuring the ACT-R architecture. Preliminary results suggest that the efficiency of this agent is acceptable for research purposes (e.g. the cognitive cycle of this agent featuring 160 rules lasted about 20 ms on average on Intel Core 2.66 GHz Duo with UT2004 server and client running [50]). Java principles and the modular architecture enable various kinds of experiments (e.g. spatial orientation, episodic memory modelling, multithreaded AI).

Different reasoning engines, such as Jadex [4], can be connected to Pogamut similarly to jACT-R. In fact, Educational scenarios described in Sec. 4.3 uses Drools rules [29] in this manner. We decided to connect jACT-R to make Pogamut 3 available for the community of computational cognitive psychology. BDI reasoners such as Jadex, which can make the platform available for traditional software agents community, can present another future goal.

4.2 Gestures Module

It is widely accepted that in order to achieve believable virtual agents, some level of gesturing and facial expressions is needed. Even though UT2004 features quite powerful animation system, gestures and facial expressions are typically expressed using animations prepared in advance (i.e. made in specialized applications). Our aim was to allow users to implement mechanisms that can compose animations of the agents' avatars as much on-the-fly as possible while depending on UT2004 as little as

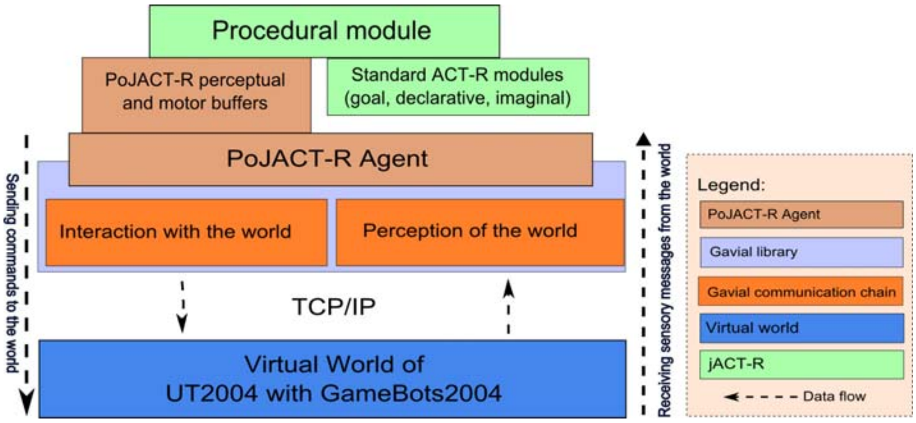

Procedural module
PoJACT-R perceptual and motor buffers
Standard ACT-R modules (goal, declarative, imaginal)
PoJACT-R Agent
Interaction with the world
Perception of the world
TCP/IP
Virtual World of UT2004 with GameBots2004
Sending commands to the world
Receiving sensory messages from the world
Legend:
PoJACT-R Agent
Gavial library
Gavial communication chain
Virtual world
jACT-R
Data flow

Fig. 5. Architecture of PojACT-R. Compare this figure with Fig. 3.

possible; in other words, to allow users to control UT2004 avatars directly from Pogamut 3. To this end, we had to find a way of defining, planning, decomposing, and transporting of animations.

Concerning planning, we decided to use Behaviour Markup Language (BML) [32]. BML is intended to become "a framework and an XML description language for controlling the communicative human behaviour of embodied conversational agents" [32]. It provides a convenient way to create and parse sequences of behaviours, including gestures, facial animations and speech, though in our case, only lexicalized gestures and gaze have been implemented so far.

For definition of lexicalized gestures, we have devised our own simplistic XML-based language, which has been designed to operate easily with BML. If reasonable, BML can be bypassed and the user can supply his own implementation of a planner through a simple interface.

For decomposition of animations the MPEG-4 with its "atomic animations" was chosen. Examples of such animations are "arm flexing," "arm twisting," or "arm abduction." These can be decomposed furthermore into lower level skeletal animations, which are then handled by Unreal Engine and composed into the final result.

Communication with the virtual world is carried using a slightly modified FBA bit-stream format specified in MPEG4 [14]. This bit-stream is quite efficient and lowers the amount of bits transferred to a bare minimum. In order to achieve even less traffic, we have decided to use a variable frame rate (instead of a constant one). On the other hand, concerning efficiency of the transfer, the limitation is the TCP protocol, which increases the traffic. Had we used RTP, the traffic would have been lower. We used TCP for prototyping purposes; in future, we plan to switch to RTP.

The work is detailed in [35] where also an example avatar is presented.

4.3 ALMA Integration

Emotion modelling is a multi-faced theme with many possible benefits for virtual agents. Emotions can affect decision making, expressive part of agents' behaviour or just serve the agent as another source of information when interacting with humans. All of these points can increase virtual agents' believability.

Therefore we decided to incorporate the ALMA emotion model [20] into Pogamut 3. ALMA is based on OCC theory of emotions [34] and provides our agents with three distinct types of affects: *emotions* for short term affects, *mood* for medium term affects and *personality* for long-term affects. ALMA is coded in Java and offers a GUI where the model can be configured easily.

ALMA integration works as follows. First, a user defines a set of events that should cause emotional responses. These events will be detected in the environment by Pogamut's methods. Second, whenever one of these events occurs in the environment, it will be appraised by a set of ALMA emotion variables and sent as an input to ALMA. Third, ALMA will process the appraised event and generate a change of affective variables.

The ALMA-to-Pogamut 3 binding was already used in Project Emohawk [5] to provide virtual agents with affects. Project Emohawk features a simple scenario, taking place in small town called UnrealVille. The virtual agents (a boy and two girls) are able to invite each other to the cinema, escort each other home or go for a walk to the park. The agents can like or dislike other agents, get angry with them or even fall in love. The analysis confirmed that the ALMA-to-Pogamut 3 binding is stable and can be used for virtual agents emotion modelling.

4.4 Educational Scenarios

Educational scenarios (ES) present a framework aimed at easy authoring of short didactic scenes. The current prototype implements a scenario for exercising orientation in urban areas. The target audience of this prototype is primary school students. The students are challenged in a game-like world with tasks ranging from following a path presented on a 2D map in a 3D visualization of a city, trough marking the player's 3D location on a 2D map, to estimating direction to the player's home. However, the ES framework can be used in a more general way.

In the content creation phase, ES provides an Eclipse IDE plug-in for specifying areas and points of interest in a 2D map of a city (e.g. street crossings, the player's home etc.). These landmarks can be then referenced from a script describing behaviour of the whole scenario. Drools rules engine [29] is used as the underlying scripting language for this purpose. A simple domain specific language resembling restricted natural English has been defined on top of the Drools rules, which makes the process of scenario creation accessible even for non-expert programmers.

ES also extends the core functionality of Pogamut 3 with a set of dialog widgets visualized in the UT2004, making it possible to run a dialog system within the game in run time. This dialog system can be used independently of the plug-in for specifying landmarks and areas on the map.

The work is detailed in [47].

5 Related Work

With respect to the Pogamut's objective, we will break down related work into three categories; a) tools for teaching students basics of development of virtual agents and virtual storytelling applications, b) tools supporting research on gaming agents' AI, c) general multi-agent platforms. Concerning Point (a), many tools for authoring 3D graphics exist, as opposed to tools for teaching programming of behaviour for virtual agents. Several authoring toolkits emerged from the cauldron of the virtual storytelling community in recent years [reviewed in 36], but they tend to provide very limited or no support for teachers and learners, many of them are poorly documented and some of them are not downloadable. To our knowledge, with respect to learning programming behaviour for virtual agents, there are only two tools (besides Pogamut) that support education explicitly and whose dissemination to schools has been documented: Netlogo [49] and Alice [15]. Netlogo is an excellent entry-level tool for building simple agents and running social simulations, but it does not allow for the creation of agents with complex behaviour and possesses no complex 3D presentation environment. Alice's main objective is the introduction of basic programming concepts like loop statements or object-oriented programming. The complexity of the built-in 3D environment is not comparable to current game engines and the environment is only a mean to achieve Alice's educational goal.

Concerning Point (b), there are several tools for the development of behaviour of human-like agents inside commercial game engines. Table 1 lists these tools along with their game engines and programming languages.

All of these tools except the Pogamut 3, UT3 .Net Bots and RIT's GameBots are using rather old game engines as their simulation environment. Quake 2 was released in 1998 and Unreal Tournament 99 in 1999. In comparison, Unreal Tournament 2004 used by Pogamut 3 gives better graphical experience. However, Unreal Tournament 3 (released in 2007) used by UT3.Net Bots is superior to all other game engines used by the listed tools. However, there is a trade-off: UT3 requires PCs with better performing hardware (i.e. graphic cards and CPUs), which may not be available, e.g. if the tool is used at high-schools for workshops.

Table 1. A list of open source tools for development of computer game agents. The year of the last release/update is based on web information, we have not contacted the authors. Thus, this information is not guaranteed.

Name	Game	Language	Last release	GUI	Note
F.E.A.R [16]	Quake 2	C++	2005	None	
Quagents [45]	Quake 2	C++	2005	None	Prolog and Matlab integration
QASE [23]	Quake 2	Java	2009	None	Matlab integration
Original Gamebots [1]	UT99	--	2001	None	Just a network protocol
RIT's Gamebots [37]	UT03, UT04	--	2007	Simple visualizer	Just a network protocol
JavaBots [28]	UT99	Java	2002	Simple	
UT3 .Net Bots [46]	UT3	C#	2008	Simple level map	
Pogamut 3 [3]	UT04	Java	2009	Netbeans plugin	POSH, BML, ACT-R

All of these frameworks are extensible but none of them besides Pogamut and Quagents comes with any 3rd party decision making system already connected. On the other hand, several tools offer a Matlab integration.

Finally, all implementations of pure Gamebots are merely communication protocols. They provide neither any feature of GaviaLib, nor a complex integrated development environment offered by Pogamut 3.

Concerning (c), the question is what is the relation of general multi-agent (MAS) platforms, such as Jade [43], to Pogamut 3 and GaviaLib. The point is that MAS platforms are typically not used in "agents in videogames" research; why is this? Our answer on this question should not be understood as a definite claim but rather as a statement aiming to motivate a further discussion on this topic. Our opinion is that MAS platforms are basically something different than toolkits like Pogamut; they have been built for different purpose. If one wants to use Jade in "agents in videogames" research, one would need to create a wrapper-agent that would encapsulate the whole virtual environment (e.g. UT2004) and that would provide communication interface to agents that would represent reasoners for in-game agents.

In other words, GameBots would be replaced by the wrapper-agent and the client-server architecture of a Pogamut-like toolkit by the MAS architecture. But why should anyone do this? On the one hand, all the work that we and others have done to implement GameBots would need to be done again in implementing the wrapper-agent, and all the work behind GaviaLib would need to be done again at the side of reasoning-agents. Actually, this reimplementation in a MAS platform may even bring a disadvantage. As GameBots communicate via a text-based protocol, the parsing is fast. If one wants to exploit FIPA-like messages [18], the parsing overhead would increase.

On the other hand, there are many features of MAS platforms without obvious use for game agents, for instance yellow pages. The whole "service-oriented" view of MAS seems to be needless for game agents. We see one possible advantage of a MAS platform in that the agents can communicate each other with directly. Each Pogamut agent can communicate with other agents in game only via the virtual environment. MAS platforms offer direct agent-agent communication channels, by-passing the environment. The trouble with this advantage is that game designers would often want to have all the communication directed via the environment for the reasons of plausibility. Still, direct agent-agent channel may be useful in more complicated games with many agents, e.g. in tactical games where the agents are required by designers to communicate each other with (e.g. solders via a walkie-talkie).

To summarize, Pogamut's main contributions are: full integration with Netbeans IDE, connection to the relatively new game engine, binding for ACT-R, POSH, BML, and ALMA, and continuing support and development of the platform.

6 Conclusion and Future Prospect

The Pogamut 3 toolkit provides many general solutions for developers of videogame agents. The important facet of Pogamut 3 is that it has been designed not only for advanced researchers, but also for newcomers. Its applicability for beginners has been proved by using it as a toolkit for teaching in a university and a high-school course on programming virtual agents [7, 9].

The current release [3] of Pogamut has also several limitations. Most notably, its integration with UT2004 and the support provided by GaviaLib classes make it most suitable for first-person shooter AI projects; this suits some, but is unsuitable for others. Although the ACT-R integration [50] is now available for cognitive psychologists and Educational scenarios [47] make Pogamut 3 partly applicable for the domain of serious games, features that would help with general virtual agents and general gaming AI development and education are limited.

To partly address this issue, we are now extending Pogamut 3 by a generic educational storytelling scenario with non-violent graphics [8], which utilizes the ALMA model [20]. Pogamut has been being connected to the Defcon game [27] and we are considering a Virtual Battle Space 2 (VBS2) [5] connection, which would feature the industry standard HLA interface (IEEE 1516) [26]. The creation of a general GaviaLib–HLA bridge would make it possible to connect even more environments with minimal effort.

Acknowledgments. Extension of several parts of this work was partially supported by the project CZ.2.17/3.1.00/31162 that is financed by the European Social Fund, the Budget of the Czech Republic and the Municipality of Prague. Other parts were partially supported by the Program “Information Society” under project 1ET100300517, by the grant 201/09/H057, and by the research project MSM0021620838 of the Ministry of Education of the Czech Republic.

References

1. Adobbati, R., Marshall, A.N., Scholer, A., Tejada, S.: Gamebots: A 3d virtual world test-bed for multi-agent research. In: Proc. of the 2nd Int. Workshop on Infrastructure for Agents, MAS, and Scalable MAS, Montreal, Canada (2001)
2. Anderson, J.R.: How can the human mind occur in the physical universe? Oxford University Press, Oxford (2007)
3. Artificial Minds for Intelligent Systems, Research Group: The Pogamut platform. Charles University in Prague, http://artemis.ms.mff.cuni.cz/pogamut (27.9.2009)
4. Braubach, L., Pokahr, A.: Jadex: BDI Agent System, http://jadex.informatik.uni-hamburg.de (27.9.2009)
5. Bída, M.: Artificial emotions in computer games. Diploma thesis. Charles University in Prague, Czech Republic (2009), http://artemis.ms.mff.cuni.cz/pogamut/tiki-index.php?page=Emotional+AI+in+UT (27.9.2009)
6. Bohemia Interactive: Virtual Battle Space 2, http://virtualbattlespace.vbs2.com/ (27.9.2009)
7. Brom, C.: Curricula of the Course on Modelling Behaviour of Human and Animal-like Agents. In: Proceedings of the Frontiers in Science Education Research Conference, Famagusta, North Cyprus, pp. 71–79 (2009)
8. Brom, C., Bída, M., Gemrot, J., Kadlec, R., Plch, T.: Emohawk: Searching for a “Good” Emergent Narrative. In: Iurgel, I.A., Zagalo, N., Petta, P. (eds.) ICIDS 2009. LNCS, vol. 5915, pp. 86–91. Springer, Heidelberg (2009)
9. Brom, C., Gemrot, J., Burkert, O., Kadlec, R., Bída, M.: 3D Immersion in Virtual Agents Education. In: Spierling, U., Szilas, N. (eds.) ICIDS 2008. LNCS, vol. 5334, pp. 59–70. Springer, Heidelberg (2008)

10. Brom, C., Lukavský, J.: Episodic Memory for Human-like Agents and Human-like Agents for Episodic Memory. Technical Reports FS-08-04, Papers from the AAAI Fall Symposium Biologically Inspired Cognitive Architectures, Westin Arlington, VA, USA, pp. 42–47. AAAI Press, Menlo Park (2008)
11. Bryson, J.J.: Intelligence by Design: Principles of Modularity and Coordination for Engineering Complex Adaptive Agents. PhD thesis, Massachusetts Institute of Technology (2001)
12. Bungie Studios: Halo 2 (2004), `http://www.bungie.net/Projects/Halo2/default.aspx` (27.9.2009)
13. Burkert, O.: Connectionist Model of Episodic Memory for Virtual Humans. Master thesis. Charles University in Prague. Czech Republic, `http://artemis.ms.mff.cuni.cz/pogamut/tiki-index.php?page=Episodic+memory+for+virtual+agent` (27.9.2009)
14. Capin, T.K., Petajan, E., Ostermann, J.: Very low bit rate coding of virtual human animation in MPEG-4. In: IEEE International Conference on Multimedia and Expo., vol. 2, pp. 1107–1110 (2000)
15. Carnegie Mellon University: Alice (1999-2009), `http://www.alice.org/` (27.9.2009)
16. Champandard, A.: F.E.A.R. – Foundations for Genuine Game AI, `http://fear.sourceforge.net/` (27.9.2009)
17. Epic Games: Unreal Tournament 2004 (2004), `http://www.unreal.com` (27.9.2009)
18. The Foundation for Intelligent Physical Agents: Agent Communication Language Specifications, `http://www.fipa.org/repository/aclspecs.html` (27.9.2009)
19. Franklin, S., Baars, B.J., Ramamurthy, U., Ventura, M.: The role of consciousness in memory. Brains, Mind, Media 1, 1–38 (2005)
20. Gebhard, P.: ALMA – A Layered Model of Affect. In: Proc. of the 4th Int. Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005), Utrecht, pp. 29–36 (2005)
21. Gemrot, J., Kadlec, R., Bida, M., Burkert, O., Pibil, R., Havlicek, J., Zemcak, L., Simlovic, J., Vansa, R., Stolba, M., Plch, T., Brom, C.: Pogamut 3 Can Assist Developers in Building AI for Their Videogame Agents. In: Dignum, F., et al. (eds.) Agents for Games and Simulations. LNCS (LNAI), vol. 5920, pp. 1–15. Springer, Heidelberg (2009)
22. Gemrot, J., Brom, C., Kadlec, R., Bída, M., Burkert, O., Zemčák, M., Píbil, R., Plch, T.: Pogamut 3 – Virtual Humans Made Simple. In: Gray, J., Nefti-Meziani, S. (eds.) Advances in Cognitive Systems. IET Publisher (in press)
23. Gorman, B., Fredriksson, M., Humphrys, M.: The QASE API - An Integrated Platform for AI Research and Education Through First-Person Computer Games. International Journal of Intelligent Games and Simulations 4(2) (2007), `http://ut3bots.codeplex.com/` (27.9.2009)
24. Harrison, A.: jACT-R project, `http://www.jactr.org/` (27.9.2009)
25. idSoftware: Quake III Arena (1999), `http://www.idsoftware.com/games/quake/quake3-arena/` (27.9.2009)
26. IEEE: IEEE 1516, High Level Architecture (HLA) (2001), `http://www.ieee.org`
27. Introversion: Defcon, `http://www.introversion.co.uk/defcon/` (4.3.2009)
28. JavaBots, `http://utbot.sourceforge.net/` (27.9.2009)
29. JBoss community: Drools project, `http://www.jboss.org/drools/` (27.9.2009)
30. Jilk, D.J., Lebiere, C., O'Reilly, R.C., Anderson, J.R.: SAL: An explicitly pluralistic cognitive architecture. Journal of Experimental and Theoretical Artificial Intelligence 20(3), 197–218 (2008)
31. Kadlec, R.: Evolution of intelligent agent behaviour in computer games. Masters thesis. Charles University in Prague, Czech Republic (2008), `http://artemis.ms.mff.cuni.cz/main/papers/GeneticBots_MSc_Kadlec.pdf` (27.9.2009)

32. Mindmakers: Behavior Markup Language (BML), Draft 1.0 (2009), http://wiki.mindmakers.org/projects:bml:draft1.0 (27.9.2009)
33. Newell, A.: Unified Theories of Cognition. Harward University Press, Cambridge (1990)
34. Ortony, A., Clore, G.L., Collins, A.: The cognitive structure of emotions. Cambridge University Press, Cambridge (1988)
35. Píbil, R.: Gestures and Facial Expressions of Virtual Humans in 3D Worlds. Bachelor thesis. Charles University in Prague, Czech Republic (in Czech) (2009), http://artemis.ms.mff.cuni.cz/pogamut/tiki-index.php?page=Gestures+and+Facial+expressions+-+BML (27.9.2009)
36. Pizzi, D., Cavazza, M.: From Debugging to Authoring: Adapting Productivity Tools to Narrative Content Description. In: Spierling, U., Szilas, N. (eds.) ICIDS 2008. LNCS, vol. 5334, pp. 285–296. Springer, Heidelberg (2008)
37. Rochester Institute of Technology: GameBots branch (2005), http://www.cs.rit.edu/~jdb/gamebots/ (26.9.2009)
38. Small, R.K.: Agent Smith: a real-time game-playing agent for interactive dynamic games. In: Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation, Atlanta, GA, USA, pp. 1839–1842 (2008)
39. Štolba, M.: Decision rules for project Pogamut 2. Bachelor thesis. Charles University in Prague, Czech Republic (in Czech) (2008)
40. Sun Microsystems: JMX, http://java.sun.com/javase/6/docs/technotes/guides/jmx/ (27.9.2009)
41. Sun Microsystems: Netbeans IDE, http://www.netbeans.org (27.9.2009)
42. Tence, F., Buche, C.: Automatable evaluation method oriented toward behaviour believability for video games. In: International Conference on Intelligent Games and Simulation (GAME-ON 2008), pp. 39–43 (2008)
43. Telecom Italia Lab: Jade – Java Agent DEvelopment Framework, http://jade.tilab.com/ (27.9.2009)
44. University of Michigan: Soar, http://sitemaker.umich.edu/soar/home (27.9.2009)
45. University of Rochester: Quagents: Quake Agents, http://www.cs.rochester.edu/research/quagents/ (27.9.2009)
46. UT3 .Net Bots project, http://ut3bots.codeplex.com (27.9.2009)
47. Vansa, R.: Educational scenarios in the Pogamut project. Bachelor thesis. Charles University in Prague, Czech Republic (2009), http://artemis.ms.mff.cuni.cz/pogamut/tiki-index.php?page=Educational+scenarios (27.9.2009)
48. Vilhjalmsson, H., Cantelmo, N., Cassell, J., Chafai, N., Kipp, M., Kopp, S., et al.: The Behavior Markup Language: Recent Developments and Challenges. In: Pelachaud, C., Martin, J.-C., André, E., Chollet, G., Karpouzis, K., Pelé, D. (eds.) IVA 2007. LNCS (LNAI), vol. 4722, pp. 99–111. Springer, Heidelberg (2007)
49. Wilensky, U.: NetLogo. Center for Connected Learning and Computer-Based Modeling, Northwestern University (1999), http://ccl.northwestern.edu/netlogo/ (27.9.2009)
50. Zemčák, L.: ACT-R in Pogamut, Bachelor thesis. Charles University in Prague, Czech Republic (in Czech) (2009), http://artemis.ms.mff.cuni.cz/pogamut/tiki-index.php?page=PojACT-R (27.9.2009)

Distributed Platform for Large-Scale Agent-Based Simulations

David Šišlák, Přemysl Volf, Michal Jakob, and Michal Pěchouček

Agent Technology Center, Department of Cybernetics,
Faculty of Electrical Engineering, Czech Technical University in Prague
sislakd@fel.cvut.cz

Abstract. We describe a distributed architecture for situated large-scale agent-based simulations with predominately local interactions. The approach, implemented in AglobeX Simulation platform, is based on a spatially partitioned simulated virtual environment and allocating a dedicated processing core to the environment simulation within each partition. In combination with dynamic load-balancing, such partitioning enables virtually unlimited scalability of the simulation platform. The approach has been used to extend the AgentFly air-traffic testbed to support simulation of a complete civilian air-traffic touching National Air-Space of United States. Thorough evaluation of the system has been performed, confirming that it can scale up to a very high number of complex agents operating simultaneously (thousands of aircraft) and determining the impact of different configurations of the simulation architecture on its overall performance.

1 Introduction

Agent-based simulations have recently become a popular way to model and study complex multi-actor systems, complementing the long-established system dynamics and discrete-event simulation approaches [3]. For a large number of agents, high complexity of their reasoning and/or high complexity of agent-environment interaction, a single machine might not be able to host the whole simulation. In such a case, an effective distribution schema is required that would enable the simulation to scale.

In this paper, we propose a particular distributed agent-based simulation architecture and describe AglobeX Simulation, a simulation platform implementing the architecture. The development of the platform has been motivated by the need to simulate, with great level of detail, full civilian air-traffic touching National Air-Space of the United States. The design and the implementation developed, however, provide a more universal recipe and can be applied to other large-scale (analytical) simulations requiring high simulation fidelity, scalability and maximum simulation speed.

The proposed approach is based on dynamic partitioning of the simulation of the virtual world into a variable number of dynamically-sized partitions, each handled by a dedicated environment simulation component. Thanks to the prevailing locality of agent-to-agent and agent-to-environment interactions, such partitioning allows to effectively distribute the required computation between a number of machines while keeping the communication overhead low.

The proposed architecture has been applied for the simulation of the full civilian air-traffic touching National Air-Space of United States, requiring high-fidelity simulation involving the total of 52,799 airplanes with up-to 6,500 simultaneous aircrafts operating over the whole Earth – long flights departing, arriving or flying over the United States. The AgentFly [18] system implementing the proposed architecture is used for a thorough empirical evaluation.

The paper is organized as follows. After a review of the related work in Section 2, we define the class of simulation problems addressed and state the requirements of the suitable solution in Section 3. In Section 4, we describe the proposed approach in detail, starting with the architecture of the simulation and the structure of the simulation up to the description of the core spatial partitioning idea and its dynamically re-balanced extension. In Section 5, we briefly describe the implementation of the approach utilizing the high-performance Aglobe multi-agent platform. Section 6 is the dedicated to an in-depth evaluation of the approach on the civilian air-traffic simulation domain. The effect of system load and the distribution of hardware resources among different components of the architecture are evaluated. Section 7 concludes with a summary and suggestions for future work.

2 Related Work

There are two basic approaches to agent-based simulations – *analytic simulation* and *distributed virtual environments*. The analytic simulation is typically used for high fidelity simulation of a given problem with a maximum *detail* and *accuracy* to provide results for further analysis of the problem. Analytic simulation runs typically without human interaction and thus a repeated simulation run with the same initial configuration leads to the same results. Furthermore, the relation to real-world time (referred to also as *wall clock*) is not important in analytic simulation and the simulation is therefore executed at maximum speed to deliver the results as soon as possible (this mode is also referred to as *as fast as possible* [6] mode). The simulation is synchronous and usually deterministic and can be based either on time steps or on events. The *time-stepped simulation* advances by predefined equally-sized (virtual simulation) time steps [8, 24] and new states of the simulation are computed after each such a step. The *event-driven simulation* uses events as a basis for its execution [2, 15]. Each event is scheduled with a time-stamp and the simulation framework advances to the time of the event with the earliest time-stamp and determines the new states.

The distributed virtual environments (DVEs) [23] are used to create a realistic illusion of real-world environment to a human user for purposes such as training or entertainment. Because of involvement of human users, DVEs are usually paced with wall clocks and do not produce the same results when repeated. They also usually do not support absolute synchronization and ordering of messages [12, 13, 25].

High level architecture (HLA) [5, 11] is an industry standard developed as an extension of DVE simulation. The architecture has been defined by a working group formed by the Defense Modeling and Simulation Office in order to standardize simulation development and allow inter-operability between various simulators produced by multiple vendors by using standardized simulation roles, predefined communication protocols and the data exchanging format.

The AglobeX Simulation described in this paper uses a combination of both the analytic and DVE approach. Agent's communication, internal processes and non-physical interaction with other agents is asynchronous, non-deterministic, possibly containing a human in the loop, and the implementation of these processes therefore uses the DVE approach. The simulation of the environment, situated entities and physical interactions in the virtual world is synchronous and deterministic, and the analytic approach with maximal accuracy is therefore used. The simulations built on the platform can run in the as-fast-as-possible mode preserving the accuracy of the simulation together with asynchronous and non-deterministic behavior of the agents in faster than wall clocks paced time.

As far as our primary application area of interest, i.e. air-traffic simulation is concerned, both the analytic and DVE approaches are used. The DVE are primarily used for training purposes and not directly relevant to our objectives. Analytic simulation, on the other hand, is used for the purposes of obtaining insight into various processes and phenomena related to air traffic. Numerous simulations with varying level of detail, different purposes and coverage have been developed. Agogino [1] uses agent-based simulation for incremental enhancements of the current *air-traffic management* (ATM). Krozel [10] uses the *FACET* simulation system to model air-traffic in inclement weather. Gorodetsky [7] uses an agent-based simulation for ATM within the air-space of large airports. Hwang [9] uses simulation for verification of collision avoidance algorithms. All of the above systems focus on simulating a particular subset of the overall air traffic. On the other hand, the AgentFly system built using the AglobeX Simulation platform aims to provide a model of the entire air traffic. The huge computational requirements resulting from this goal were consequently one of the driving forces behind the development of the presented distributed simulation platform.

3 Simulation Problem and Requirements

In this section, we describe the class of simulations addressed by this work and we specify the requirements on the solutions.

3.1 Simulation with Localized Interactions

We are addressing simulations involving large numbers of *situated entities*[1], both *autonomous* and *passive*, operating and interacting in a realistically modeled

[1] By situated entities we mean that they are embedded in a synthetic model of a 3D physical space.

large-scale *virtual world*. Given our motivating application, we further assume that the virtual world is an Earth-like 3D environment limited by a maximum altitude. The entities need not be present during the whole simulation but can be dynamically introduced or removed based on the simulation needs.

Each situated entity in the simulation carries a *state*, components of which can be either (i) *observable* i.e. public and external or (ii) *hidden* i.e. non-observable, private and internal. The fundamental component of the entity's observable state present in the simulation is its position and orientation in the virtual world.

The evolution of the entity's state is governed by the defined entity's *physics*, e.g. entity movement is driven by its motion dynamics typically defined by a set of differential equations, which can also refer to entity's hidden state components. Physics can be divided to *intra-entity* and *inter-entity*. The intra-entity physics capture those aspects of physical dynamics that can be fully ascribed to a single entity. Although the equations of the intra-entity physics can refer to any states in the simulated world, they only govern the states carried by its respective entity. In contrast, the inter-entity physics captures the dynamics that affects multiple entities simultaneously and which cannot be fully decomposed between them (consider e.g. the effects of a physical collision of two entities).

In addition to physical interactions, autonomous entities can also interact via communication, i.e. exchange electronic messages, and by using *sensors* to perceive their surrounding virtual environment. Communication between entities can be restricted by the inter-entity physics simulating required communication medium, e.g. wireless network. Similarly, each sensor has its capabilities defined which may restrict the observable state it can perceive only to a defined subset, typically based on the distance from the sensor's location (e.g. a limited range of a radar).

3.2 Requirements

Having defined the type of simulations addressed and the necessary terminology, we can now state the desired properties of a simulation platform:

- *High fidelity* – The simulation has to be accurate with respect to the defined physics. Simulation *level of details* (LOD) [4] techniques cannot be used to reduce the complexity of used intra- or inter-physics as this could lead to significant errors in the simulation output.
- *Scalability* – The simulation platform has to be scalable in terms of the number of simulated situated entities (both the number of simultaneously running entities and their total number) and the virtual world dimension. Distributed architecture of the simulation should allow to scale the simulation up simply by increasing the number of computing resources used for the simulation.
- *Maximum simulation speed* – The high-performance simulation platform should be able to produce results for a given simulation scenario in minimum possible time without sacrificing the accuracy of the high fidelity simulation. In other words, the speed with which the simulation is executed should not affect simulation results.

4 Approach and Architecture

We now describe the key components of the approach to large-scale simulations which we implemented within the AglobeX Simulation platform.

4.1 Situated Entities

In our approach, any entity consists of up to three subcomponents:

– *Body* – The entity's body encapsulates entity's intra-physics which governs the evolution of entity's state as a function of other, possibly external states. Typically, the body defines motion dynamics for the entity.
– *Reactive control* – The entity's reactive control contains real-time loop-back control for the entity. The loop-back control affects entity's states based on the observation of other states. Reactive control e.g. handle urgent reactive behaviors (such as avoiding an obstacle) triggered in emergency situations.
– *Deliberative control* – The entity's deliberative control contains complex algorithms employing planning, prediction and communication with other entities. The outputs of deliberative control typically feed back into and controls the entity's reactive control module. Typically, complex deliberative algorithms providing high-level entity control are invoked based on the sensing perceptions.

Due to their principal similarity in loop-back approach, the body and the reactive control are collectively referred to as entity's *state updater*.

Entities can be either *autonomous* or *passive*. Autonomous entities, termed *situated agents*, have at least one and typically both control modules. Situated agents are used to represent pro-active, goal-oriented actors in the simulation. Passive entities only posses the body and thus they represent non-autonomous objects whose behavior in the virtual world is fully defined only by their physics.

4.2 System Architecture

All components in the simulator architecture are divided into two groups, see Figure 1:

– *Environment simulation components* (ESCs) – These components are responsible for application of all entities' state updaters and also for the updates resulting from the inter-physics.
– *Deliberative control components* – The deliberative control part of each situated agent is encapsulated in a separate deliberative control component. Deliberative parts can communicate via component-to-component message transmission if this is allowed by inter-physics governing the communication medium.

Even though the deliberative control part and the state updater are decoupled in the simulator architecture, the deliberative control and the state updater of each entity is connected by a signal channel through which sensing perceptions (one way) and control commands to the reactive control (the other way) are transmitted.

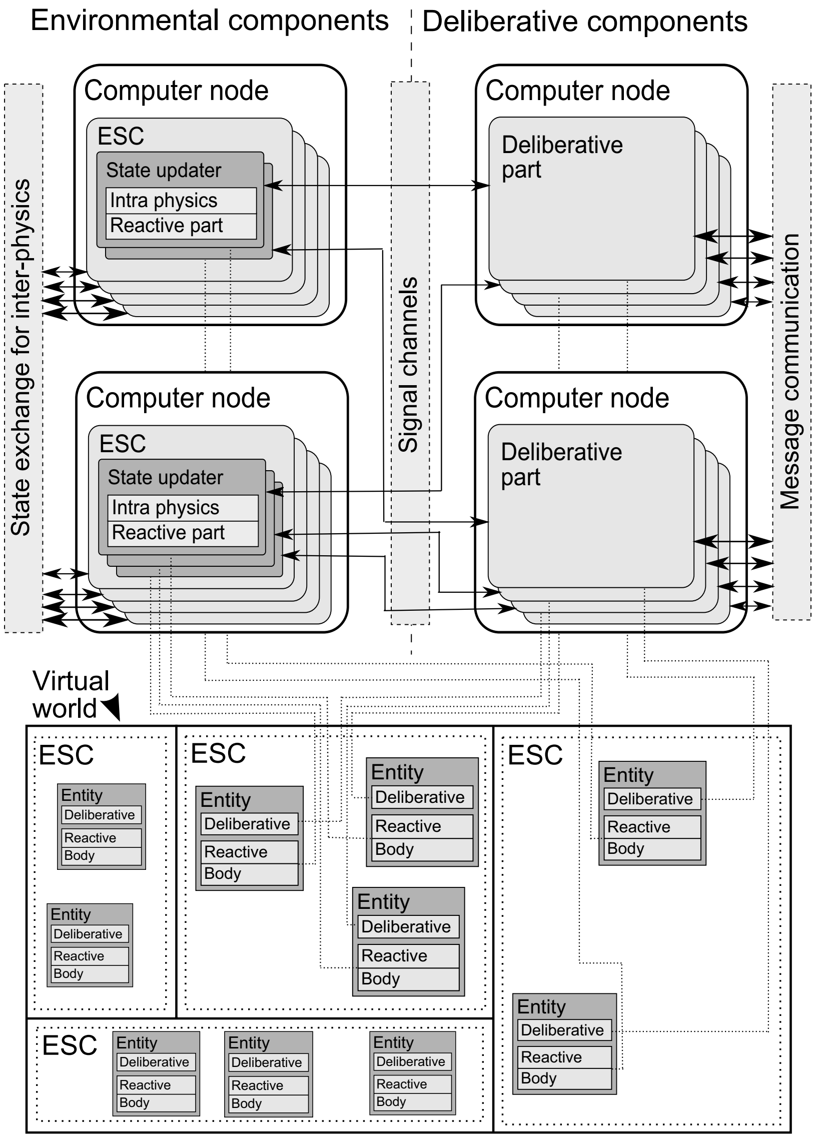


Fig. 1. Architecture overview of the distributed AglobeX Simulation platform

4.3 Time-Stepped Simulation

The presented simulation platform employs a time-stepped simulation approach [6]. The virtual simulation time driving the dynamic processes of the simulation entities is incremented uniformly by a constant time step. Defined physics and loop-back control from the reactive parts are expressed in terms of difference equations; such equations define the entity's state (after the time increment) as a function of the current state only. All these difference equations, and therefore all physics and reactive parts, have to be evaluated synchronously in order to provide consistent evolution of the virtual world. Synchronous evaluation means that states are updated only after difference equations (relevant in the given moment in time) have been evaluated.

The above virtual time steps and thus the resulting state updates can be applied regularly with respect to the external wall clock time, or in a *as-fast-as possible* manner. In the as-fast-as possible mode, the next time step is initiated just immediately after the previous one has been completed. This mode therefore allows to complete a given simulation scenario in the shortest possible time.

4.4 Distributed Simulation and Environment Partitioning

Both the environment simulation and deliberative control components can be distributed among multiple computer nodes during the simulation. The whole virtual simulation world is spatially divided into a number of partitions. The partitions are mutually disjoint except for small overlapping areas around partitions' boundaries. Each partition has a uniquely assigned environment simulation component (ESC) which is responsible for updating states (i.e applying of relevant difference equations) corresponding to all entities located within the partition. The application of intra-physics may require exchange of state information between multiple partitions, in case the entities it affects are not located within the same partition.

In contrast with the state updaters, entity's deliberative control modules are deployed in the simulation grid (computer nodes) irrespective of the partitions and corresponding entity's position in the virtual world.

The above virtual world partitioning implies that whenever the location of an entity changes so that the entity moves (spatially) to a new partition, the entity's state updater needs to be migrated to ESC responsible for the new partition. In contrast, the relatively heavy-weight deliberative control modules are never moved between computer nodes. The signal channel between the entity's reactive control and its deliberative control module is transparently reconnected and it is guaranteed that no signal transmission is lost during the migration of a state updater between two ESCs. The small overlapping areas around partition boundaries introduced above are used to suppress oscillatory migration of state updaters between neighboring ESCs in the case when the corresponding entity is moving very close along partition borders.

4.5 Simulation Cycle

The consistent evolution of the whole virtual world is achieved by coordinated operation of all ESCs. Each simulation cycle is decomposed into the following phases:

1. *Evaluation of state updaters* – During this phase, ESCs determine new state values as a result of difference equations defined in intra-physics and reactive control modules of respective entities.

2. *Evaluation of inter-physics* – In this phase, ESCs exchange state information required to calculate state updates controlled by inter-physics (and therefore possibly crossing partition boundaries)
3. *Migration of state updaters* – Entities whose location after the state update no longer belongs to the partition corresponding to their hosting ESCs are migrated to the appropriate ESCs.
4. *Application of new states* – From this moment, all pre-computed new state values are considered as current ones.

The next phase of the simulation cycle is invoked only once the current phase is completed by all ESCs.

4.6 Dynamic Load-Balancing

In order to provide maximum performance throughout the whole simulation, the proposed architecture implements load-balancing. Load-balancing related to both environment simulation components (through dynamic re-partitioning) and deliberative control parts (through balanced startup deployment) is supported.

The world decomposition to partitions is not fixed during the simulation but is dynamically updated depending on each partition's host's load. Based on the time required for processing of the first two phases of the simulation cycle (see above) by each ESCs, the world is re-partitioned so that the number of entities belonging to partitions is proportional to the measured times. ESC performing the first two phases faster will be assigned a larger area with more situated entities and thus more state updaters and vice versa. If there are no situated entities in the simulation, partitions are equally distributed covering the whole virtual world.

The above described re-partitioning is performed before the migration of state updaters (phase 3 of the simulation cycle). After re-partitioning, all state updaters which no longer belong to their currently assigned ESC partition are subsequently migrated to the appropriate ESC. Re-partitioning is not performed during each simulation cycle but only if the difference in state update evaluation times by all ESCs exceed a pre-defined threshold. The re-partitioning is also triggered whenever a new computer node is allocated to environment simulation.

The load of computer nodes running deliberative components is balanced only when new deliberative components are instantiated. Deliberative parts of newly introduced entities are spread proportionally among computer nodes according to individual nodes' average load in several previous simulation cycles, expressed as the sum of each node's idle time over those simulation cycles. The computer node with the highest idle time will receive more deliberative control modules and vice versa. There is no dynamic re-balancing of already running deliberative control modules because it would introduce issues with component-to-component addressing and proper conversation protocol consistency during the migration.

5 Implementation

The distributed simulation architecture described in the previous section has been implemented on top of the AGlobe multi-agent platform [21] and is part of the the AgentFly [18] multi-agent system for air-traffic control and planning.

AGlobe is a multi-agent middleware supporting agent encapsulation, effective agent-to-agent communication, high-throughput message passing, message topicking, yellow pages services, agent full migration, but also agent life-cycle management. AGlobe is well suited for scalable multi-agent simulation as it supports integration of simulation components. AGlobe requires limited computational resources and its operation is very efficient. AGlobe also facilitates modeling communication inaccessibility and unreliability in ad-hoc networking environment. AGlobe has been used for number of applications and agent system prototypes in the area of air traffic control, car electronic modeling, design process simulation or network intrusion detection.

In the designed distribution simulation platform, each ESC and each deliberative control module are encapsulated as an AGlobe software agent. Beside agents that represent ESCs and deliberative control modules, there are also other agents which take care of simulation maintenance and coordination among distributed system.

AgentFly [18] is a complex multi-agent system for modeling and simulation of air-traffic, involving both manned and unmanned aircraft. The framework provides models of aircraft physics, advanced trajectory path planning algorithms [22], multi-layer collision avoidance implementing both cooperative and non-cooperative collision avoidance [20], integration interfaces for external data sources, high-fidelity 3D visualization and data collectors for post-run analysis tools. AgentFly has been integrated with the algorithms for planning tactical missions, information collection and surveillance [19]. The system scalability has been dramatically enhanced, by integration of the presented distributed simulation platform, in order to answer commercial traffic simulation requirements.

6 Evaluation

We have evaluated the proposed distributed simulation architecture on a large-scale simulation of civilian air-traffic (see the domain description below). The objective of the experiments were two-fold: (i) to understand the performance of the simulation platform with constant computing resources under varying level of system load, i.e. the size/complexity of the simulation scenario (Section 6.4), and (ii) to understand the performance of the simulation system under varying distribution of the available resource between key platform components (Section 6.5).

6.1 U.S. National Airspace Traffic Domain

The proposed simulation platform has been deployed for the simulation of civilian air-traffic within U.S. National Airspace (U.S. NAS). The resulting airspace

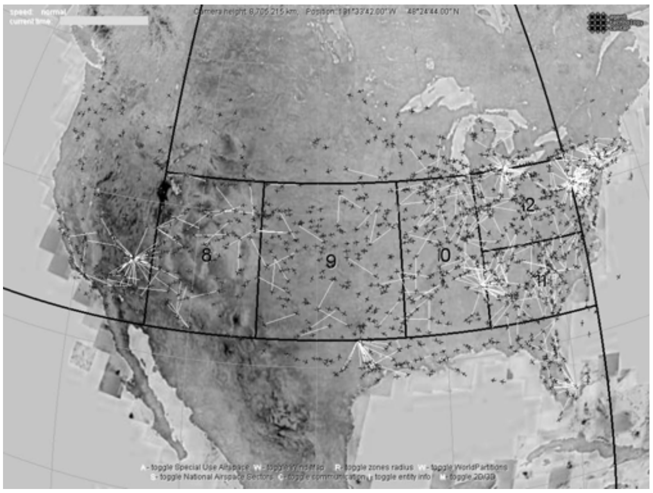


Fig. 2. Airplanes over U.S. NAS and world partitioning

simulator is used for studying concepts for the Next Generation Air Transportation Systems (NGATS) [14] in cooperation with the Federal Aviation Administration (FAA).

Each civilian flight operated under Instrumented Flight Rules (IFR) [16] is represented as a situated entity within the simulation. One simulation run typically covers one day and simulates operation of all civilian IFR flights which traverse, fully or partially, U.S. NAS. Even though the area of the study is limited to U.S. NAS, some effects external to U.S. NAS are also studied, covering almost the whole Earth for long-distance flights. The overall number of simulated airplanes is 52,799 (up to 6,500 are simultaneous) corresponding to real air-traffic for a particular day in 2007, see Figure 2. Depending on the airplane type, each entity uses a precise flight model based on the BADA airplane performance models [17].

The distribution of the number of airplanes during the simulation day time is depicted in Figure 3. Note that there are significant peaks every half an hour of the simulation time; these peaks lead to extreme load of the system with more than 300 airplanes created at one simulation cycle in the highest peak of the day. For each such a newly created airplane, a corresponding situated entity needs to be spawned and the computation-intensive flight trajectory planning and flight control algorithms executed.

The simulation cycle is 12 seconds long, which is suitable for this domain. A scalable simulation platform is required because the system is used for analysis of concepts related to the predicted growth of air traffic in the upcoming decades (i.e. a simulation of the expected density of traffic in 5, 10, 15, and 20 years).

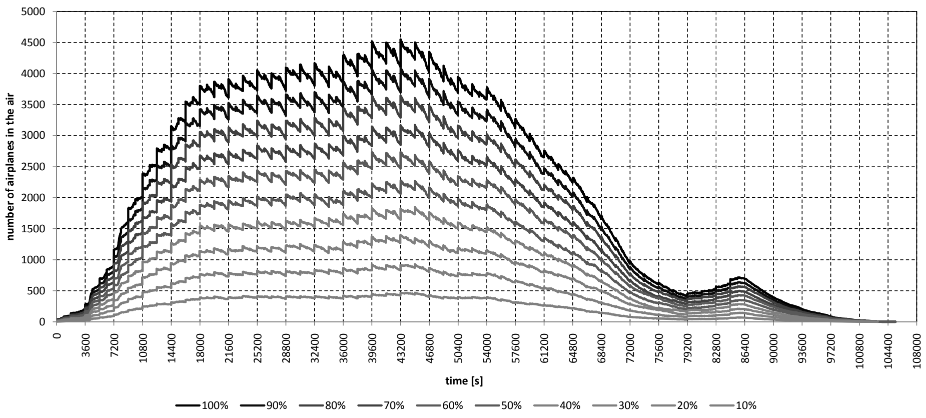


Fig. 3. The number of simultaneous airplanes during the simulation

6.2 Metrics

The *run-time* is the total real (wall clock) time needed for the simulation multiplied by the number of processor cores used during the simulation.

The *normalized airplane* is a fictive airplane that is flying during the whole simulation from its start to the end. The number of airplanes is changing during the simulation, thus the mean of the number of airplanes is used as the number of normalized airplanes running during the whole simulation. This number is used to normalize run-time to study requirements per each situated entity.

6.3 Testing Configuration

The configuration contained a set of several heterogenous (both in terms of the operating system and hardware) computer nodes. During experiments, nodes were dedicated for ESCs or deliberative controllers, see computer node allocation in Figure 4:

The testing configuration consisted of 9 cores with 9GB RAM for ESCs and 11 cores with 15GB RAM for the flight control integrated in deliberative controllers. All computers were interconnected through a 1 Gbit Ethernet network (connection through a single switch) using the TCP/IP protocol. All systems were running Sun Java 64-bit 1.6.0_14 environment. All empirical values are averaged from five independent measurements with identical initial conditions, e.g. a freshly launched JVMs, no allocated memory etc.

6.4 Overall System Load

The objective of the first experiment was to explore how the speed of the simulation changes with the changing level of system load. The experiment was performed using all computer nodes. The variations in system load were introduced by varying the number of airplanes simulated, ranging from 10% up to 100% of all air-traffic (with 10% step).

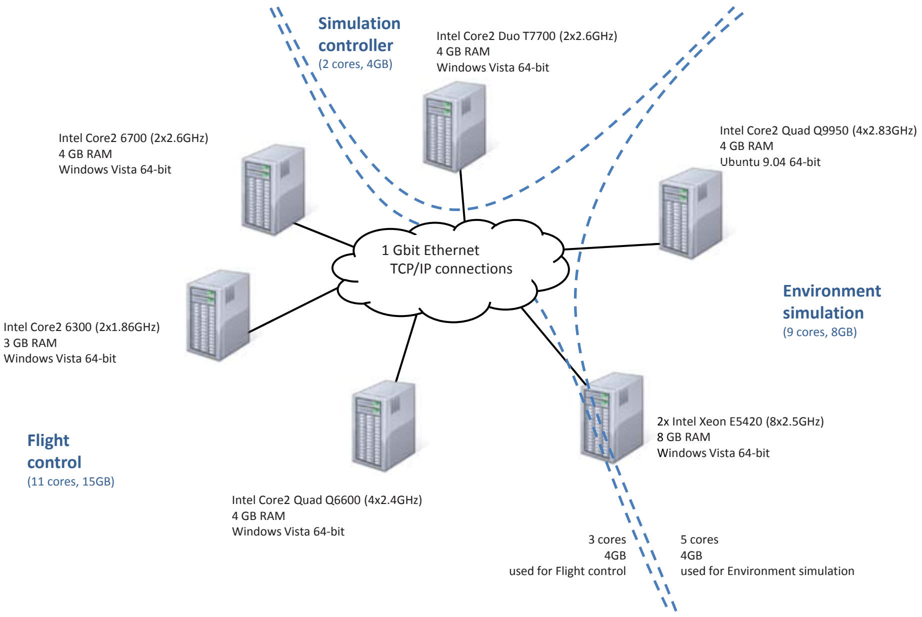


Fig. 4. Computer nodes allocation during experiment: *environment simulation* – hosts environmental components, *flight control* – hosts all deliberative control parts and *simulation controller* – other components for simulation control and input/output interfacing to the simulation

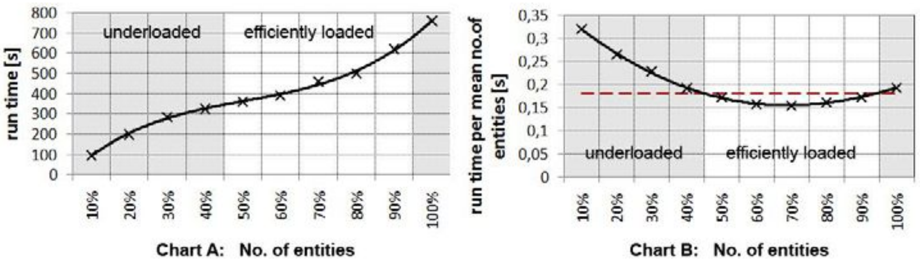


Fig. 5. Effect of the increasing number of flights simulated with fixed hardware resources, measured in terms of the total duration of the simulation (Chart A) and run-time per normalized airplane (Chart B)

The aggregated results of the experiment are depicted in Figure 5 showing the correspondence between the total duration of the simulation and the number of flights. Chart A shows the maximum duration is less than 13 minutes for 100% of flights. Chart B shows run-time needed to simulate one normalized airplane.

The experiment demonstrates properties and effectiveness of the distributed simulation. The measured characteristics can be divided into three regions depending on the system load (the number of airplanes): (i) underloaded, (ii) optimally loaded and (ii) overloaded simulation. The regions can be determined by the specified threshold, a dashed horizontal line at 0.18 s in Figure 5, Chart B.

The upper-left part of the plot above the threshold corresponds to the underloaded simulation, the part bellow the threshold is the optimally loaded simulation and the upper-right part above the plot is the overloaded simulation.

Underloaded Simulation. At low levels of system load (few situated entities), the simulation is underloaded and thus inefficient. The hardware resources cannot be effectively used because the load of the simulation is too low (e.g. the simulation running 5 airplanes cannot effectively use 20 processor cores), but there is an overhead given by the maintenance of distributed architecture which depends both on the number of processors cores and the number of airplanes in the system. The shape of the load line in Figure 5, Chart B is concave as the overhead of the distributed simulation is split among the increasing number of airplanes. The underloaded simulation leads to higher run-time per normalized airplane.

Optimally loaded Simulation. In the optimally loaded configuration, the simulation is able to use all available hardware resources effectively. This section begins at the moment when all hardware can be used and ends when some components start to be overloaded. Within the optimally loaded region, the increasing load of the simulation is distributed regularly among all resources, leading to almost linear increase of the simulation run-time.

Overloaded Simulation. Moving towards the highest load of the fixed computer configuration, the hardware resources become overloaded. The overload has three main reasons:

CPU overload. The simulation management is designed to run the simulation as fast as possible considering the CPU load. If the CPU performance is not sufficient, the simulation continuously slows down until the whole simulation almost stops.

Memory overload. Once the simulation requires more than available physical memory, this results in higher probability of page faults and causes swapping which leads to decrease of the simulation performance.

Network overload. Despite the distributed design which inherently aims to minimize the network traffic, the network can get overloaded. If the network is not able to transmit all traffic required by the simulation especially related to the environment simulation, the simulation is paused and waits for their delivery.

In the experiments, usually both CPU and memory resources were overloaded. Thus the simulation gradually slows down at the beginning of the overload and with higher load it slows down quickly.

6.5 Varying Resource Distribution

The aim of the second group of experiments was to asses the performance impact of different distributions of available hardware resources between environmental and deliberative control components in the simulation. Both experiments used a simulation load consisting from 40% of the flights.

The first experiment was performed using the full test hardware configuration for the deliberative control parts (see Figure 4) and varying hardware configuration for the environmental components. The second experiment was performed using the full test hardware configuration for the environmental components and varying hardware configuration for the deliberative control parts.

The results for the first and second experiment are depicted in Figures 6 and 7, respectively. The left plots in both Figures show how the simulation run-time decreases with adding hardware resources. The right plots in both Figures show changes in normalized run-time.

The first experiment (Figure 6) shows the dependency between the overall performance and the number of computational cores assigned to the environment components and thus resulting in different numbers of partitions in the simulation. The left region of the charts (one, two and three partitions) shows a significant speed-up in both the simulation run-time and normalized run-time. This is because the small number of partitions in the simulation are overloaded. Adding more hardware and thus more partitions (three, four and five partitions) corresponds to the optimally-utilized simulation. The simulation run-time is still reduced, however, the normalized run-time stagnates. This is caused by rising demands for partition-to-partition coordination. Adding even more resources brings almost no speed-up in the simulation run-time and even results in a decreasing normalized run-time. The reason is that the simulation configuration is unable to fully-utilize the newly introduced partitions in comparison to increasing overhead for partition-to-partition coordination.

The second experiment(Figure 7) shows a similar picture for the dependency of the simulation run-time on the number of available hardware resources for deliberative controllers. In the left part (two, three and four cores), it shows a

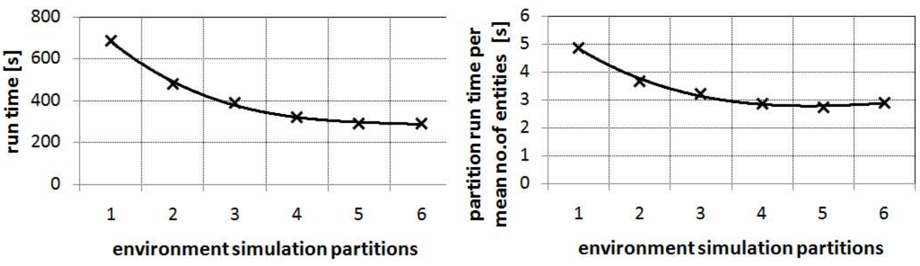


Fig. 6. The effect of changing the hardware resources allocated to environment components while keeping the resources allocated to the rest of the system constant; the effect measured in terms of the total simulation run-time (left) and total run-time per normalized airplane (right)

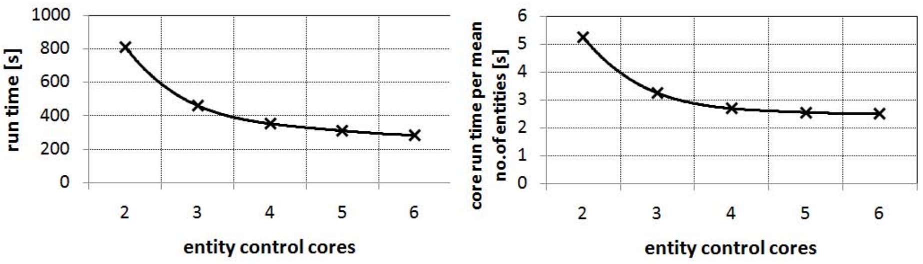


Fig. 7. The effect of changing the hardware resources allocated to deliberative control parts while keeping the resources allocated to the rest of the system constant; the effect measured in terms of the total simulation run-time (left) and simulation run-time per normalized airplane (right)

significant speed-up for the same reasons as in the previous case. The different property is for more hardware resources (four, five and six cores). In such a case, the simulation stays optimally loaded, the simulation run-time decreases and the normalized run-time remains constant. The reason is that deliberative controllers are independent and do not require any additional coordination. All computational resources can be therefore used for parallel computation. At a certain moment, we can identify that there is no use in adding more hardware resources for the deliberative controllers as the bottle neck is in the environment part for the given simulation configuration.

The simulation performance is maximized by the proper distribution of available resources among environmental components and deliberative controllers. Under-resourcing of any part leads to a very significant decrease of the overall system performance. Over-resourcing of the deliberative controllers has no effect on performance, but it can help when the load to the system increases. Over-resourcing of the environment controllers can cause decrease of the overall performance due to an additional coordination overhead induced by a higher number of partitions.

7 Conclusion

We have investigated the problem of efficient simulation of a huge number of situated agents operating in large-scale virtual worlds. A distributed, fully scalable approach with automated load balancing has been proposed to address the problem. The approach efficiently exploits the predominantly local nature of interactions in situated agent-based simulations and uses it to efficiently partition the virtual world and distribute it over multiple computer nodes. The approach has been implemented in a AglobeX Simulation agent-based simulation platform based on AgentFly test-bed, its specific application to the air-traffic domain, and evaluated on the simulation of the complete civilian air-traffic touching the U.S. NAS involving the total of 52,799 flights with up to 6,500 airplanes simulated at one time. The evaluation confirms the ability of the approach to handle very large-scale agent-based simulations and highlights the need to carefully assign hardware resources to individual components of the simulation platform.

Acknowledgement

The work has been supported by the Federal Aviation Administration (FAA) under project number DTFACT-08-C-00033 and by Czech Ministry of Education under grant number 6840770038. The underlying AGENTFLY system was supported by the Air Force Office of Scientific Research, Air Force Material Command, USAF, under grant number FA8655-06-1-3073. The U.S. Government is authorized to reproduce and distribute reprints for Government purpose notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the author and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of the Federal Aviation Administration, the Air Force Office of Scientific Research or the U.S. Government.

References

[1] Agogino, A., Tumer, K.: Regulating air traffic flow with coupled agents. In: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems, vol. 2, pp. 535–542. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC (2008)

[2] Banks, J., Carson, J.S., Nelson, B.L., Nicol, D.M.: Discrete-event system simulation. Prentice Hall, Upper Saddle River (2001)

[3] Borshchev, A., Filippov, A.: From system dynamics and discrete event to practical agent based modeling: Reasons, techniques, tools. In: Proceedings of the 22nd International Conference of the System Dynamics Society, pp. 25–29 (2004)

[4] Brom, C., Šerý, O., Poch, T.: Simulation level of detail for virtual humans. In: Pelachaud, C., Martin, J.-C., André, E., Chollet, G., Karpouzis, K., Pelé, D. (eds.) IVA 2007. LNCS (LNAI), vol. 4722, pp. 1–14. Springer, Heidelberg (2007)

[5] Dahmann, J.S., Fujimoto, R.M., Weatherly, R.M.: The department of defense high level architecture. In: Proceedings of the 29th conference on Winter simulation, pp. 142–149. IEEE Computer Society, Washington (1997)

[6] Fujimoto, R.M.: Parallel and Distribution Simulation Systems. John Wiley & Sons, Inc., New York (1999)

[7] Gorodetsky, V., Karsaev, O., Samoylov, V., Skormin, V.: Multi-Agent Technology for Air Traffic Control and Incident Management in Airport Airspace. In: Proceedings of the International Workshop Agents in Traffic and Transportation, Portugal, pp. 119–125 (2008)

[8] Guo, Y., Gong, W., Towsley, D., Amherst, M.A.: Time-stepped hybrid simulation (TSHS) for large scale networks. In: IEEE INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings, vol. 2 (2000)

[9] Hwang, I., Kim, J., Tomlin, C.: Protocol-based conflict resolution for air traffic control. Air Traffic Control Quarterly 15(1), 1–34 (2007)

[10] Krozel, J., Doble, N.: Simulation of the National Airspace System in Inclement Weather. In: AIAA Modeling and Simulation Technologies Conference, Hilton Head, SC (2007)
[11] Kuhl, F., Weatherly, R., Dahmann, J.: Creating computer simulation systems: an introduction to the high level architecture. Prentice Hall PTR, Upper Saddle River (1999)
[12] Lui, J.C.S., Chan, M.F.: An efficient partitioning algorithm for distributed virtual environment systems. IEEE Trans. Parallel Distrib. Syst. 13(3), 193–211 (2002)
[13] Morillo, P., Ordufia, J.M., Duato, J.: A scalable synchronization technique for distributed virtual environments based on networked-server architectures. In: 2006 International Conference on Parallel Processing Workshops, ICPP 2006 Workshops, p. 8 (2006)
[14] National Research Council Panel on Human Factors in Air Traffic Control Automation. The Future of Air Traffic Control: Human Factors and Automation. National Academy Press (1998)
[15] Nicol, D.M., Yan, G.: Discrete event fluid modeling of background TCP traffic. ACM Transactions on Modeling and Computer Simulation (TOMACS) 14(3), 211–250 (2004)
[16] Nolan, M.S.: Fundamentals of Air Traffic Control, 4th edn. Thomson Brooks/Cole, Belmont (2004)
[17] Nuic, A., Poinsot, C., Iagaru, M.G., Gallo, E., Navarro, F.A., Querejeta, C.: Advanced Aircraft Performance Modelling for ATM: Enhancements to the BADA Model. In: Beitrag zur 24th Digital Avioncs System Conference. DASC, Washington (2005)
[18] Pěchouček, M., Šišlák, D.: Agent-based approach to free-flight planning, control, and simulation. IEEE Intelligent Systems 24(1) (2009)
[19] Semsch, E., Jakob, M., Pavlíček, D., Pěchouček, M., Šišlák, D.: Autonomous uav surveillance in complex urban environments. In: McGann, C., Smith, D.E., Likhachev, M., Marthi, B. (eds.) Proceedings of ICAPS 2009 Workshop on Bridging the Gap Between Task and Motion Planning, Greece, September 2009, pp. 63–70 (2009)
[20] Šišlák, D., Volf, P., Komenda, A., Samek, J., Pěchouček, M.: Agent-based multilayer collision avoidance to unmanned aerial vehicles. In: Lawton, J. (ed.) Proceedings of 2007 International Conference on Integration of Knowledge Intensive Multi Agent Systems, KSCO. IEEE, Los Alamitos (2007)
[21] Šišlák, D., Rehák, M., Pěchouček, M., Rollo, M., Pavlíček, D.: Aglobe: Agent development platform with inaccessibility and mobility support. In: Unland, R., Klusch, M., Calisti, M. (eds.) Software Agent-Based Applications, Platforms and Development Kits, Berlin, pp. 21–46. Birkhauser Verlag, Basel (2005)
[22] Šišlák, D., Volf, P., Pěchouček, M.: Accelerated a* path planning. In: The Eighth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009), Budapest, Hungary (May 2009)
[23] Waters, R.C., Barrus, J.W.: The rise of shared virtual environments. IEEE Spectrum 34(3) (March 1997)
[24] Wu, Y., Gong, W.: Time-stepped simulation of queueing systems. In: Proceedings of SPIE, vol. 4367, p. 262 (2001)
[25] Zhou, S., Cai, W., Lee, B.S., Turner, S.J.: Time-space consistency in large-scale distributed virtual environments. ACM Transactions on Modeling and Computer Simulation (TOMACS) 14(1), 31–47 (2004)

Two Case Studies for Jazzyk BSM

Michael Köster, Peter Novák, David Mainzer, and Bernd Fuhrmann

Department of Informatics, Clausthal University of Technology,
Julius-Albert-Str. 4, 38678 Clausthal-Zellerfeld, Germany
{mko,pno,dm,ifbf}@tu-clausthal.de
http://cig.in.tu-clausthal.de/

Abstract. Recently, we introduced *Behavioural State Machines* (*BSM*), a novel programming framework for development of cognitive agents with *Jazzyk*, its associated programming language and interpreter. The *Jazzyk BSM* framework draws a strict distinction between *knowledge representation* and *behavioural* aspects of an agent program. *Jazzyk BSM* thus enables synergistic exploitation of heterogeneous knowledge representation technologies in a single agent, as well as offers a transparent way for embedding cognitive agents in various simulated or physical environments. This makes it a particularly suitable platform for development of simulated, as well as physically embodied cognitive agents, such as virtual agents, or non-player characters for computer games.

In this paper we report on *Jazzbot* and *Urbibot* projects, two case-studies we developed using the *Jazzyk BSM* framework in simulated environments provided by a first person shooter computer game and a physical reality simulator for mobile robotics respectively. We describe the underlying technological infrastructure of the two agent applications and provide a brief account of experiences and lessons we learned during the development.

1 Introduction

One of the long-term aims of Artificial Intelligence is to enable development of intelligent cognitive agents. I.e. such which internally model their environment, their own mental attitudes, reason about them and subsequently base their decisions regarding their future actions upon these models. Even though AI research provides a plethora of approaches for solving partial problems on the way towards this aim, we only rarely encounter approaches enabling integration of the various developed technologies. The field of agent oriented programming, and in consequence multi-agent systems programming, offers a sound theoretical basis allowing synergistic exploitation of heterogeneous AI technologies in a single agent system.

Therefore in our recent work, we introduced the theoretical framework of *Behavioural State Machines* with its associated agent oriented programming language *Jazzyk* [11,12]. *Jazzyk BSM* provides a simple, theoretically sound language for modular agent programming based on a generic computational model for reactive systems. It draws a strict distinction between the knowledge representation (KR) and behavioural aspects of an agent program and thus enables exploiting heterogeneous KR technologies in a single agent system.

To provide a proof-of-concept, as well as to further nurture our research towards a methodology of development with *Jazzyk BSM* (cf. [14] and [13]), we developed two case study applications *Jazzbot* and *Urbibot*. *Jazzbot* is a virtual bot in the simulated 3D environment of an open source first person shooter computer game *Nexuiz*. Its task is to explore a virtual building, search for certain objects in it and subsequently deliver them to the base. At the same time, *Jazzbot* is supposed to differentiate between other players present in the building and seek safety upon being attacked by an enemy player. When the danger disappears, it should return back to the activity interrupted by the attack.

Urbibot, on the other hand, was developed as a step towards programming mobile robots. It is an agent program steering a model of customized *e-Puck*, a small two-wheeled mobile robot in an environment provided by the physical robotic simulator *Webots*. Similarly to *Jazzbot*, *Urbibot* explores its environment in order to find red poles present in it. It tries to bump into each of them, while trying to avoid patrol robots policing the environment. Upon encounter with such a patrol robot, *Urbibot* runs away to finally return to the previously interrupted activity when safe again.

Both agents feature a BDI inspired architecture. While interacting with two different types of virtual bodies, a character in the game and an interface to robot hardware sensors and actuators respectively, both implementations exploit the power of non-monotonic reasoning for representation and reasoning about their beliefs and goals. We employ an interpreted object oriented programming language to enable efficient representation of and reasoning about topological structure of the environment.

After a brief introduction to the framework of *Behavioural State Machines* and its associated programming language *Jazzyk* in Section 2, Sections 3 and 4 describe respectively *Jazzbot* and *Urbibot* agents in a closer detail. Subsequently, Section 5 provides a description of the underlying technological infrastructure used in the implemented agents. Finally, a discussion of our experiences and lessons learned from the development of *Jazzbot* and *Urbibot* agents, together with an outlook to the ongoing and future work wraps up the paper in Section 6.

2 Jazzyk BSM

In [12] we introduced the framework of *Behavioural State Machines* (*BSM*). *BSM* framework draws a clear distinction between the *knowledge representation* and *behavioural* layers within an agent. It thus provides a programming system that clearly separates the programming concerns of *how to represent an agent's knowledge* about, for example, its environment and *how to encode its behaviours*. In the core of the framework is a *generic reactive computational model* inspired by Gurevich's *Abstract State Machines* [3], enabling for efficient structuring of the program code. This section briefly introduces the *BSM* framework. For the complete formal description of the *BSM* framework, see [12].

2.1 Syntax

BSM agents are collections of one or more so-called *knowledge representation modules* (KR modules), typically denoted by $\mathcal{M}$, each representing a part of the agent's knowledge base. KR modules may be used to represent and maintain various mental attitudes of an agent, such as knowledge about its environment, or its goals, intentions, obligations, etc. Transitions between states of a *BSM* result from applying so-called *mental state transformers* (*mst*), typically denoted by τ. Various types of mst's determine the behaviour that an agent can generate. A *BSM agent* consists of a set of KR modules $\mathcal{M}_1, \dots, \mathcal{M}_n$ and a mental state transformer $\mathcal{P}$, i.e. $\mathcal{A} = (\mathcal{M}_1, \dots, \mathcal{M}_n, \mathcal{P})$; the mst $\mathcal{P}$ is also called an *agent program.*

The notion of a KR module is an abstraction of a partial knowledge base of an agent. In turn, its states are to be treated as theories (i.e. sets of sentences) expressed in the KR language of the module. Formally, a KR module $\mathcal{M}_i = (\mathcal{S}_i, \mathcal{L}_i, \mathcal{Q}_i, \mathcal{U}_i)$ is characterized by a knowledge representation language $\mathcal{L}_i$, a set of states $\mathcal{S}_i \subseteq 2^{\mathcal{L}_i}$, a set of query operators $\mathcal{Q}_i$ and a set of update operators $\mathcal{U}_i$. A query operator $\models \in \mathcal{Q}_i$ is a mapping $\models : \mathcal{S}_i \times \mathcal{L}_i \rightarrow \{\top, \bot\}$. Similarly an update operator $\oplus \in \mathcal{U}_i$ is a mapping $\oplus : \mathcal{S}_i \times \mathcal{L}_i \rightarrow \mathcal{S}_i$.

Queries, typically denoted by φ, can be seen as operators of type $\models : \mathcal{S}_i \rightarrow \{\top, \bot\}$. A primitive query $\varphi = (\models\phi)$ consists of a query operator $\models \in \mathcal{Q}_i$ and a formula $\phi \in \mathcal{L}_i$ of the same KR module $\mathcal{M}_i$. Complex queries can be composed by means of conjunction $\wedge$, disjunction $\vee$ and negation $\neg$.

Mental state transformers enable transitions from one state to another. A primitive mst $\oslash\psi$, typically denoted by ρ and constructed from an update operator $\oslash \in \mathcal{U}_i$ and a formula $\psi \in \mathcal{L}_i$, refers to an update on the state of the corresponding KR module. Conditional mst's are of the form $\varphi \longrightarrow \tau$, where φ is a query and τ is a mst. Such a conditional mst makes the application of τ depend on the evaluation of φ. Syntactic constructs for combining mst's are: non-deterministic choice $|$ and sequence $\circ$.

Definition 1 (mental state transformer). *Let $\mathcal{M}_1, \dots, \mathcal{M}_n$ be KR modules of the form $\mathcal{M}_i = (\mathcal{S}_i, \mathcal{L}_i, \mathcal{Q}_i, \mathcal{U}_i)$. The set of* mental state transformers *is defined as below:*

- **skip** *is a* primitive *mst,*
- *if $\oslash \in \mathcal{U}_i$ and $\psi \in \mathcal{L}_i$, then $\oslash\psi$ is a* primitive *mst,*
- *if φ is a query, and τ is a mst, then $\varphi \longrightarrow \tau$ is a* conditional *mst,*
- *if τ and τ' are mst's, then $\tau|\tau'$ and $\tau \circ \tau'$ are mst's (*choice*, and* sequence *respectively).*

Even though it is a vital feature of the *BSM* theoretical framework, for simplicity we omit the treatment of variables in the definitions of query and update formulae above. For a full fledged description of the BSM framework consult [12].

2.2 Semantics

The *yields* calculus, summarised below after [12], specifies an update associated with executing a mental state transformer in a single step of the language interpreter. It formally defines the meaning of the state transformation induced by executing an mst in a state, i.e. a mental state transition.

Formally, a *mental state* σ of a *BSM* $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \tau)$ is a tuple $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ of KR module states $\sigma_1 \in \mathcal{S}_1, \ldots, \sigma_n \in \mathcal{S}_n$, corresponding to $\mathcal{M}_1, \ldots, \mathcal{M}_n$ respectively. $\mathcal{S} = \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$ denotes the space of all mental states over $\mathcal{A}$. A mental state can be modified by applying primitive mst's on it and query formulae can be evaluated against it. The semantic notion of truth of a query is defined through the satisfaction relation $\models$. A primitive query $\vDash\phi$ holds in a mental state $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ (written $\sigma \models (\vDash\phi)$) iff $\vDash(\phi, \sigma_i)$, otherwise we have $\sigma \not\models (\vDash\phi)$. Given the usual meaning of Boolean operators, it is straightforward to extend the query evaluation to compound query formulae. Note that evaluation of a query does not change the mental state σ.

For an mst $\oslash\psi$, we use $(\oslash, \psi)$ to denote its semantic counterpart, i.e., the corresponding *update* (state transformation). Sequential application of updates is denoted by $\bullet$, i.e. $\rho_1 \bullet \rho_2$ is an update resulting from applying ρ_1 first and then applying ρ_2. The application of an update to a mental state is defined formally below.

Definition 2 (applying an update). *The result of applying an update* $\rho = (\oslash, \psi)$ *to a state* $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ *of a* BSM $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$, *denoted by* $s \bigoplus \rho$, *is a new state* $\sigma' = \langle \sigma_1, \ldots, \sigma'_i, \ldots, \sigma_n \rangle$, *where* $\sigma'_i = \sigma_i \oslash \psi$ *and* σ_i, $\oslash$, *and* ψ *correspond to one and the same* $\mathcal{M}_i$ *of* $\mathcal{A}$. *Applying the empty update* **skip** *on the state* σ *does not change the state, i.e.* $\sigma \bigoplus \textbf{skip} = \sigma$.

Inductively, the result of applying a sequence of updates $\rho_1 \bullet \rho_2$ *is a new state* $\sigma'' = \sigma' \bigoplus \rho_2$, *where* $\sigma' = \sigma \bigoplus \rho_1$. $\sigma \overset{\rho_1 \bullet \rho_2}{\rightarrow} \sigma'' = \sigma \overset{\rho_1}{\rightarrow} \sigma' \overset{\rho_2}{\rightarrow} \sigma''$ *denotes the corresponding compound transition.*

The meaning of a mental state transformer in state σ, formally defined by the *yields* predicate below, is the update set it yields in that mental state.

Definition 3 (yields calculus). *A mental state transformer* τ *yields an* update ρ *in a state* σ, *iff* $yields(\tau, \sigma, \rho)$ *is derivable in the following calculus:*

$$\frac{\top}{yields(\textbf{skip},\sigma,\textbf{skip})} \quad \frac{\top}{yields(\oslash\psi,\sigma,(\oslash,\psi))} \quad \textit{(primitive)}$$

$$\frac{yields(\tau,\sigma,\rho),\ \sigma\models\phi}{yields(\phi\longrightarrow\tau,\sigma,\rho)} \quad \frac{yields(\tau,\sigma,\rho),\ \sigma\not\models\phi}{yields(\phi\longrightarrow\tau,\sigma,\textbf{skip})} \quad \textit{(conditional)}$$

$$\frac{yields(\tau_1,\sigma,\rho_1),\ yields(\tau_2,\sigma,\rho_2)}{yields(\tau_1|\tau_2,\sigma,\rho_1),\ yields(\tau_1|\tau_2,\sigma,\rho_2)} \quad \textit{(choice)}$$

$$\frac{yields(\tau_1,\sigma,\rho_1),\ yields(\tau_2,\sigma \bigoplus \rho_1,\rho_2)}{yields(\tau_1\circ\tau_2,\sigma,\rho_1\bullet\rho_2)} \quad \textit{(sequence)}$$

We say that τ *yields an* update set ν *in a state* σ *iff* $\nu = \{\rho | yields(\tau, \sigma, \rho)\}$.

The mst **skip** yields the update **skip**. Similarly, a primitive update mst $\oslash\psi$ yields the corresponding update $(\oslash, \psi)$. In the case the condition ϕ of a conditional mst $\phi \longrightarrow \tau$ is satisfied in the current mental state, the calculus yields one of the updates corresponding to the right hand side mst τ, otherwise the no-operation

skip update is yielded. A non-deterministic choice mst yields an update corresponding to either of its members and finally a sequential mst yields a sequence of updates corresponding to the first mst of the sequence and an update yielded by the second member of the sequence in a state resulting from application of the first update to the current mental state.

The following definition articulates the denotational semantics of the notion of mental state transformer as an encoding of a function mapping mental states of a *BSM* to updates, i.e. transitions between them.

Definition 4 (mst functional semantics). *Let $\mathcal{M}_1, \ldots, \mathcal{M}_n$ be KR modules. A mental state transformer τ encodes a function $\mathfrak{f}_\tau : \sigma \mapsto \{\rho | yields(\tau, \sigma, \rho)\}$ over the space of mental states $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle \in S_1 \times \cdots S_n$.*

Subsequently, the semantics of a *BSM* agent is defined as a set of traces in the induced transition system enabled by the *BSM* agent program.

Definition 5 (BSM semantics). *A* BSM *$\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$ can make a step from state σ to a state σ', iff $\sigma' = \sigma \bigoplus \rho$, s.t. $\rho \in \mathfrak{f}_\mathcal{P}(\sigma)$. We also say, that $\mathcal{A}$ induces a (possibly compound) transition $\sigma \xrightarrow{\rho} \sigma'$.*

A possibly infinite sequence of states $\sigma_1, \ldots, \sigma_i, \ldots$ is a run of BSM $\mathcal{A}$, iff for each $i \geq 1$, $\mathcal{A}$ induces a transition $\sigma_i \to \sigma_{i+1}$.

The semantics of an agent system characterized by a BSM $\mathcal{A}$, is a set of all runs of $\mathcal{A}$.

Additionally, we require the non-deterministic choice of a *BSM* interpreter to fulfil the *weak fairness condition*, similar to that in [9], for all the induced runs.

Condition 1 (weak fairness condition). *A computation run is weakly fair iff it is not the case that an update is always yielded from some point in time on but is never selected for execution.*

2.3 Jazzyk

Jazzyk is an interpreter of the *Jazzyk* programming language implementing the computational model of the *BSM* framework. The syntax of the *Jazzyk* language is an instantiation of the abstract mathematical syntax of the *BSM* theoretical framework. when ϕ then τ construct encodes a conditional *mst* $\phi \longrightarrow \tau$. Symbols ; and , stand for choice | and sequence $\circ$ operators respectively. To facilitate operator precedence, mental state transformers can be grouped into compound structures, blocks, using curly braces {...}.

To better support source code modularity and re-usability, *Jazzyk* interpreter integrates GNU M4[1], a state-of-the-art macro preprocessor. Macros are a powerful tool for structuring and modularizing and encapsulating the source code and writing code templates. Before feeding the *Jazzyk* agent program to the language interpreter, first all the macros are expanded. Listing 1 depicts a *Jazzyk* code snippet. For further details on the *Jazzyk* programming language and the macro preprocessor integration with *Jazzyk* interpreter, consult [12].

[1] `http://www.gnu.org/software/m4/`

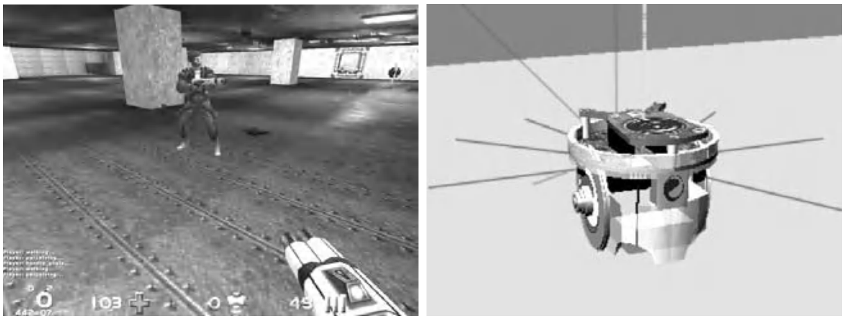


Fig. 1. Screenshots of the *Jazzbot* and *Urbibot* agents

3 Jazzbot

Jazzbot is a virtual agent embodied in a simulated 3D environment of the first-person shooter computer game *Nexuiz*[2]. It is a goal-driven BDI inspired cognitive agent developed with the *Jazzyk* language. The *Nexuiz* death-match game takes place in a virtual building containing various objects (e.g. weapons, flags or armor kits), is capable of simulating diverse terrains like solid floor, or liquid and provides a basic means for inter-player interaction. Because of its accessibility (*Nexuiz* is published under the open source GNU GPL licence), we chose the *Nexuiz* game server as the simulator for *Jazzbot* case-study, the first larger proof-of-concept application for the *Jazzyk BSM* framework. Figure 1 left depicts a screenshot of the *Jazzbot* agent acting in the simulated environment. Demonstration videos and source code can be found on the project website[3].

Jazzbot's behaviour is implemented as a *Jazzyk* program. In the experimental scenario, the bot searches for a particular item in the environment, which it then picks up and delivers to the base point. While during the search phase the agent tries to always move to unexplored segments of the environment, when it tries to deliver the item, it exploits a path planning algorithm to compute the shortest path to the base point. Hence, during the search phase, in every step the bot randomly selects a direction to move to a previously unexplored part of the building and in the case there is none such, it returns to the nearest waypoint from which an unexplored direction exists. The behaviour for environment exploration is interrupted, whenever *Jazzbot* feels under attack, i.e. an enemy player attempts to shoot at it. Upon that it triggers emergency behaviours, such as running away from the danger. After the sense of emergency fades away, it returns back to its previously performed goals of item search, or delivery.

The *Jazzbot*'s control cycle consists of three steps that are executed sequentially. Firstly, the bot reads its sensors (perception), then if necessary it deliberates about its goals, (goal commitment strategies implementation) and finally it selects an action according to its actual goals and beliefs (act). Listing 1 provides

[2] `http://www.alientrap.org/nexuiz/`
[3] `http://jazzyk.sourceforge.net/`

Listing 1. Code snippet from the *Jazzbot* agent

```
define('ACT','{
  /* The bot searches for an item, only when it does not have it */
  when ⊨_goals [{ task(search(X)) }] and not ⊨_beliefs [{ hold(X) }] then SEARCH('X');
  /* When a searched item is found, it picks it */
  when ⊨_goals [{ task(pick(X)) }] and ⊨_beliefs [{ see(X) }] then PICK('X') ;
  /* When the bot finally holds the item, it deliver it */
  when ⊨_goals [{ task(deliver(X)) }] and ⊨_beliefs [{ hold(X) }] then DELIVER('X') ;
  /* Simple behaviour triggers without guard conditions */
  when ⊨_goals [{ task(wander) }] then WALK ;
  when ⊨_goals [{ task(safety) }] then RUN_AWAY ;
  when ⊨_goals [{ task(communicate) }] then SOCIALIZE
}')
```

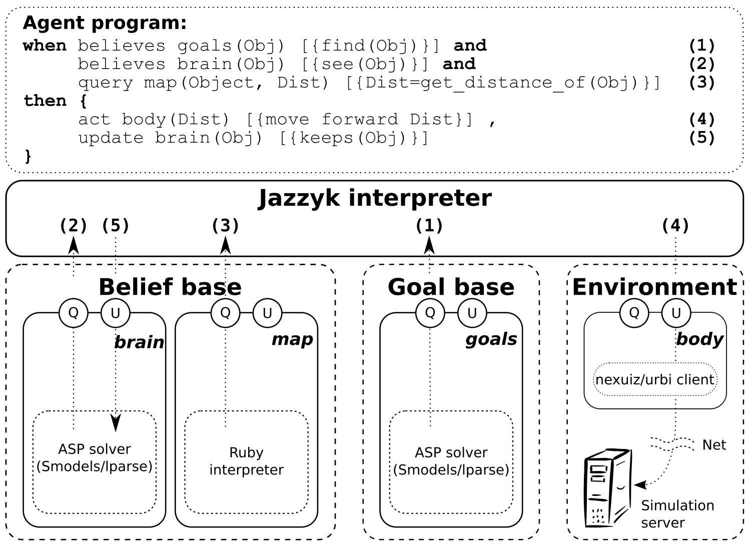
an example code implementing selection of goal oriented behaviours, realized as parametrized macros, triggered by *Jazzbot*'s goals. While the bot simply triggers behaviours for walking around, danger aversion and social behaviour, execution of behaviours finally leading to getting an item are guarded by belief conditions.

The Figure 2 provides an overview of the *Jazzbot*'s architecture. The agent features a belief base consisting of two KR modules for representation of agent's actual beliefs and storing the map of the environment, a goal base encoding interrelationships between various agent's declarative, performance and maintenance goals and finally the module interfacing the bot with the simulated environment.

JzNexuiz KR module (cf. Subsection 5.4), the *Jazzbot*'s interface to the environment, the body, provides the bot with capabilities for sensing and acting in the virtual world. The bot can move forward, backward, it can turn, or shoot. Additionally, the *Jazzbot* is equiped with several sensors: GPS, sonar, 3D compass and an object recognition sensor. The module communicates over the network with the *Nexuiz* game server and thus provides an interface of a pure client side *Nexuiz* bot, i.e. the bot can access only a subset of the perceptual information a human player would have available.

The *Jazzbot*'s *belief base* is composed of two modules: *JzASP* (cf. Subsection 5.1) and *JzRuby* (cf. Subsection 5.2). While the first one integrates an *Answer Set Programming* [2] (*ASP*) solver *Smodels* [18] and contains a logic program reflecting agent's beliefs about itself, the environment, objects in it and other players, the second, based on an interpreted object oriented programming language *Ruby*, stores the map of the agent's environment.

The *Jazzbot*'s *goal base* is again an *ASP* logic program representing agent's current goals and their interdependencies. Goals can be either of a declarative (*goals-to-be*), or performative nature (*goals-to-do*, or *tasks*). In *Jazzbot* agent implementation, each *goal-to-do* activates one, or more *tasks*, which in turn trigger, one or more corresponding behaviours the agent is supposed to execute. On the ground of holding certain beliefs, the agent is also allowed to adopt new, or drop goals which are either satisfied, irrelevant, or subjectively recognized as impossible to achieve. The agent thus implements goal *commitment strategies*. We explore the details of the programming methodology employed in the *Jazzbot* project in [14]. The Section 5 below provides details on the implementation of the *Jazzbot*'s KR modules.



```
Agent program:
when believes goals(Obj) [{find(Obj)}] and                          (1)
     believes brain(Obj) [{see(Obj)}] and                           (2)
     query map(Object, Dist) [{Dist=get_distance_of(Obj)}]          (3)
then {
     act body(Dist) [{move forward Dist}] ,                         (4)
     update brain(Obj) [{keeps(Obj)}]                               (5)
}
```

Fig. 2. Internal architecture of *Jazzbot* and *Urbibot* agents

4 Urbibot

Urbibot [4] is the second case-study developed as a step towards applications of the *Jazzyk BSM* framework in the mobile robotics domain. It is a robot exploring a maze where it searches for red poles and then tries to kick them down while at the same time avoiding patrols policing the space. *Urbibot* is embodied as an *e-Puck*[4], a small educational mobile robot simulated in *Webots*[5] [10], a robotics oriented physical world simulator. The robot is steered using *URBI*[6], a highly flexible and modular robotic programming platform based on event-based programming model. The main motivation for using *URBI* is the direct transferability of the developed agent program from simulator to the real robot.

Similarly to the *Jazzbot*, the overall agent design is inspired by the BDI architecture and reuses parts of the code developed for *Jazzbot*. In turn, except for using *JzASP* KR module to represent agent's beliefs about itself, *Urbibot* features similar agent architecture as the one depicted in the Figure 2 for the *Jazzbot* agent. *Urbibot*'s beliefs comprise exclusively information about the map. The interface with the simulator environment is provided by the *JzUrbi* KR module (see Subsection 5.3).

As already noted above, *Urbibot*'s behaviour is similar to that of *Jazzbot* agent. However instead of controlling the agent's body with rather discrete commands, such as `move forward`, or `turn left`, *Urbibot*'s *URBI* allows a more sophisticated control by directly accessing the robot's actuators, which in the case of the

[4] `http://www.e-puck.org/`

[5] `http://www.cyberbotics.com/`

[6] `http://www.gostai.com/`

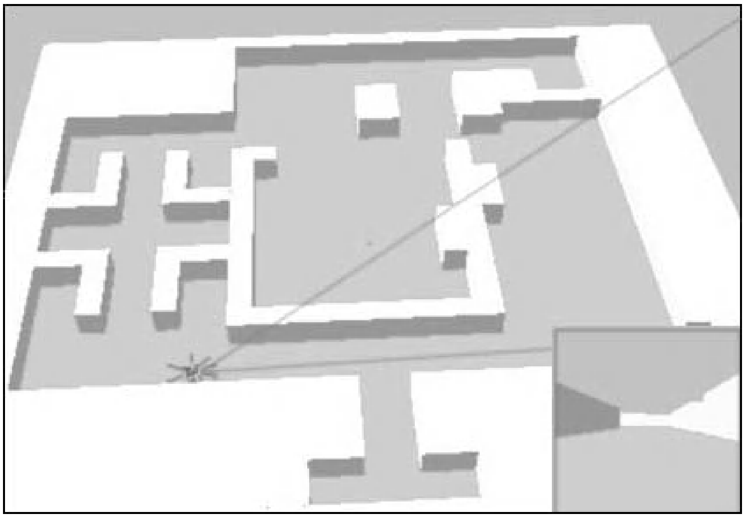


Fig. 3. *Urbibot* exploring the simulated environment. The lower right corner provides the current snapshot of the *Urbibot*'s camera perception.

e-Puck robot, are only it's two wheels. The robot features a mounted camera, a directional distance sensor and an additional GPS sensor (our customization of the original *e-Puck* robot). In the *JzRuby* module, the robot analyzes the camera image stream and by joining it with the output of the distance and GPS sensors it constructs a 2D map of the environment. Upon encountering a patrol robot, *Urbibot* calculates an approximation of the space the patrol robot can see, and subsequently tries to navigate out of this area as quickly as possible. Again, the details on the implementation of the *Urbibot*'s KR modules can be found later in the Section 5. Figures 1 (right) and 3 depicts a screenshot of the *Urbibot* agent and the maze environment with the *Urbibot* acting in it. Furthermore, demonstration videos and source code are provided in the corresponding section of the *Jazzyk* project website[3].

5 Modules

The *Jazzyk Software Development Kit (Jazzyk SDK)* provides a C++ interface from which each KR module plug-in has to be derived. Basically, the class defines five methods: `initialize`, `finalize`, `cycle`, `query` and `update`. It is possible to define multiple `query` and `update` methods corresponding to KR module's query and update operators. These methods then define the plug-in's interface to the *Jazzyk* interpreter. While initialize, finalize and cycle are mainly used for initialization, shutdown and maintenance of the module the query and update provide means for modification of the stored knowledge base.

Below we describe KR module's we employed in development of the *Jazzbot* and *Urbibot* case studies. We introduce two modules facilitating agent's knowledge representation *JzASP* and *JzRuby* followed by description of two modules, *JzUrbi* and *JzNexuiz*, interfacing the agents with their respective environments.

5.1 JzASP

In [5] we presented the *JzASP* module. It integrates *Lparse* [17] and *Smodels* [18], an *Answer Set Programming* grounder and solver respectively. The stored knowledge base thus consists of a logic program in the syntax of *AnsProlog** [2] (*Prolog* style syntax) and can be accessed by two query methods sure_believes and poss_believes and two update methods add and del allowing for retrieval and modification of the stored knowledge base. Internally, the *JzASP* module processes the program by passing it to the *Lparse* library and subsequently let's *Smodels* solver to compute the program's answer sets.

The two query methods sure_believes and poss_believes check whether the query formula, an *AnsProlog** term, is contained in all the computed answer sets, or there exists at least a single answer set containing it respectively. Before the query formula is processed by the module, all the free variables occurring in it are substituted by their valuations and subsequently, the query method attempts matching the remaining free variables with a term from a computed answer set.

The update interface methods add and del provide a means to assert, or retract a clause (a fact, or a rule) to/from the stored knowledge base. The variable substitution treatment is similar to that in processing query formulae.

While the *Jazzbot* agent employs the *JzASP* for both, reasoning about its beliefs regarding its environment, other agents and its own body state, as well as to represent and reason about its goals, the *Urbibot* agent employs the module only to treat its goal base. Using the power of non-monotonic reasoning, in particular the default negation, to reason about agent's goals turned out to be advantageous and led to an elegant encoding of interrelations between various goals. We elaborate more on the technique used in [14].

5.2 JzRuby

The *JzRuby* module, detailed description in [4], integrates the interpreted object oriented scripting language *Ruby*[7]. The KR module interface methods for initialization and finalization as well as the query and update routines are able to process plain *Ruby* programs as argument formulae (query/update). *Jazzyk* variables are treated as global variables in the *Ruby* interpreter's memory space. Query invocations of the single query method return the truth value of the code invocation within the *Ruby* interpreter, i.e. provided the code execution yields a value other than 0, the KR module returns $\top$ and $\bot$ otherwise. The single update method of the KR module simply executes the provided update formula, a plain *Ruby* code chunk.

To represent the *Jazzbot*'s information about the topology of its environment, the agent uses a *circle-based waypoint graph* (*CWG*) [16] to generate the map of

[7] `http://www.ruby-lang.org/`

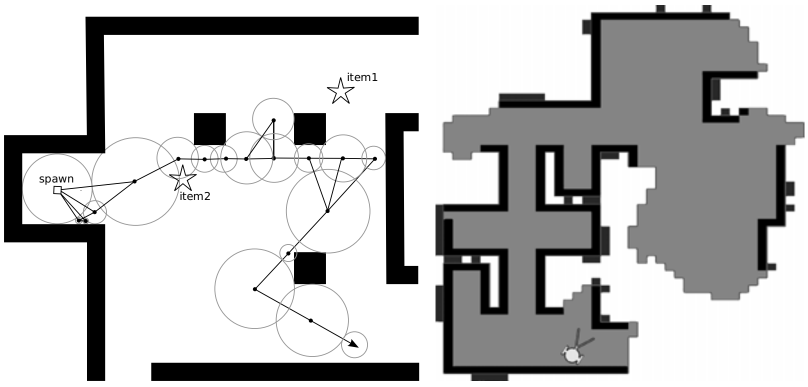


Fig. 4. Environment maps representation in *Jazzbot* and *Urbibot*

its environment. *CWG*'s are an improved version of *waypoint graphs*, extended with a radius for each waypoint. The radius is determined by the distance between the avatar and the nearest obstacle. This technique ensures, especially in big rooms or open spaces, a smaller number of nodes and connections within the graph what in turn speeds up the path search algorithm. Figure 4 (left) shows a graphical representation of the *CWG* for a sample walk of the *Jazzbot* agent from the spawn point to the point marked by the arrow.

Additionally, each waypoint stores a list of objects present within its range as well as about walls touching it and information about unexplored directions, i.e. such in which there's no connection to another waypoint, nor a wall. By employing a breadth-first graph search algorithm, the agent can compute the shortest path to a particular object, or a position.

The *CWG* graph is constructed by the agent so that in each step it determines whether its current absolute position corresponds to some known waypoint, and if not, it turns around in 60° steps and by checking its distance sensor, it determines the nearest obstacle around. Subsequently, the newly added waypoint is incorporated into the *CWG* by connecting it to all the other waypoints with which it overlaps and all the perceived objects together with all the directions in which the agent can see a wall are stored with it.

While similarly to the *Jazzbot*, the *Urbibot* uses the *JzRuby* KR module for representing its environment too, the map representation approach differs. *Urbibot* represents the map of its environment as a 2D grid, where in each cell it stores an information about its content, such as `unknown`, `free`, `wall`, `patrol`, `avoid` or `pole`. A new row or column is added to the grid when the robot reaches the edge of the known region. While the agent continuously adds new information to cells, the map becomes more and more precise. Furthermore, the *Urbibot* employs the A^* path planning algorithm to compute the shortest path between the current position and a position to go, be it an unexplored cell, the nearest safe cell a patrol robot cannot see, or a cell containing a pole which it tries to kick down. Also, the *Urbibot* uses the *JzRuby* module to algorithmically process the sensory input from its mounted camera and the distance sensors.

5.3 JzUrbi

In order to interface a *Jazzyk* program with the robot's body, the *JzUrbi* KR module [4] integrates the *URBI* programming language interpreter. It connects over TCP/IP to an *URBI* server on the simulator's side, or with the *URBI* robot controller that controls the robot's body.

The single query method query provide the agent program with the sensor information from the body. *Jazzyk* variables are treated the same way as in *JzRuby*, i.e. as global variables of the underlying *URBI* interpreter. Similarly, the single update method update simply sends the provided update formula, an *URBI* program, to the *URBI* server.

The *Urbibot* connects over the *JzUrbi* module with the *URBI* server running in the *Webots* simulator and thereby steers the *e-Puck* robot, extended with a GPS sensor. In the particular case of the *Urbibot*, the sensory input accesses the following sensors: camera, distance sensor, GPS, touch-sensor and a light-sensor. Together with the update interface steering the *Urbibot*'s two wheels, these two methods provide the basic interface for *e-Puck*'s control.

5.4 JzNexuiz

Finally, the *JzNexuiz* KR module, invented in [8], facilitates *Jazzbot*'s interaction with the *Nexuiz* game environment. By the means of a single query interface method sense and a single update method act, it enables control of the *Jazzbot*'s avatar body in the virtual building. The query method provides access to several sensors of GPS, sonar, 3D compass and object recognition. The update method allows issuing commands to the avatar's body, such as move, turn, jump, use, attack or say.

Technically, the module connects over TCP/IP with a *Nexuiz* server and thus provides an interface of a pure client side *Nexuiz* bot. The consequence of this setup is that the *Jazzbot* agent can access only a strict subset of the perceptual information a human player would have. The *Jazzbot* plug-in integrates a stripped down and customized *Nexuiz* client. In turn, the bot's actions are implemented as the corresponding key strokes of a virtual player.

Figure 5 depicts the syntax accepted by the plug-in's query and update interface methods sense and act. Each query formula starts with the name of the accessed virtual sensor device followed by the corresponding arguments being either constants, or variables facilitating retrieval of information from the environment. Similarly, the update formulas consist of the action to be executed by the avatar followed by a list of arguments specifying the command parameters.

The truth values of query formula evaluation depends on the sensory input retrieved from the environment. In the case the query evaluates to true ($\top$), the additional information about e.g. the distance of an obstacle, or the reading of the body health sensor, is stored in provided free variables.

```
nex_query  ::= sensor (constant | variable)+
nex_update ::= action (constant | variable)+
sensor     ::= sen_const | variable
sen_const  ::= 'body' | 'liquid' | 'ground' | 'gps' | 'compass' |
               'sonar' | 'map' | 'eye' | 'listen'
action     ::= act_const | variable
act_const  ::= 'move' | 'turn' | 'jump' | 'use' | 'attack' | 'say'
```

Fig. 5. *JzNexuiz* EBNF

6 Experiences and Conclusion

The two case-studies described in this paper served us most importantly as a vehicle to nurture and pragmatically drive our research towards a methodology for using an agent oriented programming language exploiting strengths of heterogeneous KR technologies in a single cognitive agent system. For further details concerning the methodology consult [14]. As an important side effect, we collected experiences with programming BDI inspired virtual cognitive agents for computer games and simulated environments, as well.

As in the long run we aim at development of autonomous robots, in both cases the virtual agents had to be running autonomously and independently from the simulator of the environment. This choice had a strong impact on the design of the agents w.r.t. the action execution model and the model of perception. In both described applications, the agents are remotely connecting to the simulated environment in which they execute actions in an asynchronous manner, i.e. they can only indirectly observe the effects (success/failure) of their actions through later perceptions. As far as the model of perception is concerned, unlike other game bots, *Jazzbot* is a pure client side bot, i.e. the amount of information it can perceive is a strict subset of the information provided to the game client used by human players. Hence, the *Jazzbot* agent cannot take advantage of additional information, such as the global topology of the environment, or information about objects in distant parts of the environment, which are accessible to the majority of other bots available for first-person shooter games. In the case of *Urbibot*, the simulator provides only perceptions accessible to the models of robot's sensors. In our case these are most importantly a camera, a directional distance sensor and global positioning, hence the available information is, similarly to *Jazzbot*, only local, incomplete and noisy.

As both implemented agents are running independently from the simulation engine and execute their actions in an asynchronous manner, their efficiency is only loosely coupled to the simulation platform speed. In our experiments, the speed of agent's reactions was reasonable w.r.t. task the bots were supposed to execute. However, especially in the case of *Jazzbot*, due to deficiencies on the side of sensors, such as missing camera rendering the complete scene the bot can see, *Jazzbot* in its present incarnation cannot match the reaction speed of advanced human players in a peer-2-peer match.

Since, the agents store their internal state in the application domain specific KR modules, the control model of *Jazzyk BSM* framework results in agents which can instantly change the focus of their attention w.r.t. an observed change of the context in the environment. The goal orientedness of agent's behaviours emerges from the coupling between behaviour triggers and agent's attitudes modeled in its components [14]. This turned out to be of a particular advantage when a quick reaction to interruptions, such as an encounter of an enemy agent, or a patrol, was needed. On the other hand, because of the open plug-in architecture of the *Jazzyk BSM* framework, we were able to quickly prototype and experiment with various approaches to knowledge representation and reasoning, as well as various models of interaction with the environment.

Our research project follows the spirit of [7], where Laird and van Lent argue that approaches for programming intelligent agents should be tested in realistic and sophisticated environments of modern computer games. *Jazzbot* project thus follows in footsteps of their *SOAR QuakeBot* [6].

Another relevant project, *Gamebots* [1], provides a general purpose interface to a first-person shooter game *Unreal Tournament*[8]. *Gamebots*' approach is however server side, i.e. the virtual agent is provided with much more information than a human player has, what was not in the spirit of our aim to emulate mobile robots in a virtual environment. Why we did not pick the *Gamebots* framework for our project was also the fact, that it is specific to the commercially available game *Unreal Tournament* and since 2002, the project does not seem to be further maintained.

To our knowledge, our work on the *Jazzbot* and *Urbibot* case-studies is novel in the sense that it seems to be the first efficient application of non-monotonic reasoning framework of *ASP* in a highly dynamic domain of simulated robotics, or a first-person shooter computer game. Even though, to our knowledge the first attempt by Provetti et al. [15] uses *ASP* for planning and action selection in the context of the *Quake 3 Arena*[9] game, authors note that their bot could not recalculate its plans rapidly enough since each computation required up to 7 seconds in a standard setup [15]. Thus, in comparison to *Jazzbot* or *Urbibot*, their agent was capable to react to events occurring in the environment only to a lesser extend, because both their action selection and planning was in *ASP*.

Similarly, to our knowledge, the transparent integration of various KR technologies such as a declarative, logic based technology for representing agent's beliefs and goals, an object oriented language for storing the topological information about the environment together with a generic reactive control model of the agent program in the *Jazzyk BSM* framework is unique. In consequence, the *Jazzyk BSM* framework shows a lot of potential for further experimentation with synergies of exploiting various AI technologies in cognitive agent systems, especially in the attractive domain of virtual agents and autonomous non-player characters, for computer games. Yet, we believe, more experimentation is needed

[8] `http://www.unrealtournament3.com/`

[9] `http://www.idsoftware.com/`

to explore the limits and deficiencies of our approach in the domain of simulated, as well as physical reality embodied robotics.

References

1. Adobbati, R., Marshall, A.N., Scholer, A., Tejada, S., Kaminka, G.A., Schaffer, S., Sollitto, C.: Gamebots: A 3D Virtual World Test-Bed For Multi-Agent Research. In: Proceedings of the Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS (2001)
2. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press, Cambridge (2003)
3. Börger, E., Stärk, R.F.: Abstract State Machines. A Method for High-Level System Design and Analysis. Springer, Heidelberg (2003)
4. Fuhrmann, B.: Implementierung eines URBI- und Rubymoduls für Jazzyk zur Entwicklung von Robotern. Master's thesis, Clausthal University of Technology (2009)
5. Köster, M.: Implementierung eines autonomen Agenten in einer simulierten 3D-Umgebung - Wissensrepräsentation. Master's thesis, Clausthal University of Technology (2008)
6. Laird, J.E.: It knows what you're going to do: adding anticipation to a Quakebot. In: Proceedings of the fifth international conference on Autonomous agents, pp. 385–392. ACM, New York (2001)
7. Laird, J.E., van Lent, M.: Human-level AI's killer application: Interactive computer games. AI Magazine 22(2), 15–26 (2001)
8. Mainzer, D.: Implementierung eines autonomen Agenten in einer simulierten 3D-Umgebung - Interaktion mit der Umwelt. Master's thesis, Clausthal University of Technology (2008)
9. Manna, Z., Pnueli, A.: The temporal logic of reactive and concurrent systems. Springer, New York (1992)
10. Michel, O.: Webots: Symbiosis between virtual and real mobile robots. In: Heudin, J.-C. (ed.) VW 1998. LNCS (LNAI), vol. 1434, pp. 254–263. Springer, Heidelberg (1998)
11. Novák, P.: Behavioural State Machines: programming modular agents. In: AAAI 2008 Spring Symposium: Architectures for Intelligent Theory-Based Agents, AITA 2008, March 26-28, pp. 49–54 (2008)
12. Novák, P.: Jazzyk: A programming language for hybrid agents with heterogeneous knowledge representations. In: Proceedings of the Sixth International Workshop on Programming Multi-Agent Systems, May 2008, pp. 143–158 (2008)
13. Novák, P., Jamroga, W.: Code patterns for agent-oriented programming. In: AAMAS (to appear, 2009)
14. Novák, P., Köster, M.: Designing goal-oriented reactive behaviours. In: Proceedings of the 6th International Cognitive Robotics Workshop, CogRob 2008, Patras, Greece, July 21-22, pp. 24–31 (2008)
15. Padovani, L., Provetti, A.: Qsmodels: Asp planning in interactive gaming environment. In: Alferes, J.J., Leite, J. (eds.) JELIA 2004. LNCS (LNAI), vol. 3229, pp. 689–692. Springer, Heidelberg (2004)
16. Rabin, S.: AI Game Programming Wisdom 2. Charles River Media (2004)
17. Syrjänen, T.: Lparse 1.0 User's Manual. University of Helsinki, Finland (2000)
18. Syrjänen, T., Niemelä, I.: The Smodels System. In: Eiter, T., Faber, W., Truszczyński, M. (eds.) LPNMR 2001. LNCS (LNAI), vol. 2173, pp. 434–438. Springer, Heidelberg (2001)

A Teamwork Infrastructure for Computer Games with Real-Time Requirements

Ivan Medeiros Monteiro and Luis Otavio Alvares

Universidade Federal do Rio Grande do Sul, Instituto de Informática,
Av. Bento Goncalves, 9500, Porto Alegre - RS, Brazil
Tel.: +55 51 3316 6843; Fax: +55 51 3316 7308
{immonteiro,alvares}@inf.ufrgs.br

Abstract. Although there are many researches in teamwork, the development of agent teams for complex environments still present many challenges, especially if these environments have real-time requirements. Many tools have been developed, but there is no silver bullet, and the most general tools have serious problems with the real-time requirements. This paper introduces a new proxy-based tool, based on *Joint Intentions*, to help agents to be a teammate in partially observable, dynamic and stochastic environments with real-time requirements. Validation experiments with the computer game Unreal Tournament 2004 have demonstrated impressive results.

1 Introduction

In computer games most effort has been done to improve graphic aspects, but very little has been done considering the behavioral aspects, in order to reduce the distance to the human behavior [3]. The situation of team-based games is even worse, because bots[1] usually do not assume coherent roles inside the team. Thus, although there is progress on individual behavior, not much has been done for social behavior in computer games.

Teamwork[2] has emerged as the paradigm for coordinating cooperative agents in dynamic environments and it has been shown to be capable of leading to flexible and robust behavior [29]. However, there is no general solution to build agent teams and each new domain may have new requirements. It still remains a challenge for teamwork applications in which the domain is highly dynamic and requires a short response time.

Teamwork has been applied to many domains, such as rescue disaster [19], military combat [7] [9], robocup soccer [10] [15], and collaboration between humans and agents [20]. Flexible teamwork, promoted by the explicit team model, that defines commitments and responsibilities for teammate, allows a robust coordination, keeping coherence even with individuals faults and unpredictable changes in the environment [26].

[1] A character controlled by an artificial agent.
[2] A cooperative effort realized by teammates to achieve a goal.

Some theories have been proposed to formalize the behavior of a group of agents to act as a team. Two of these theories have important results for real problems. They are *Joint Intentions* [14] and *Shared Plans* [5]. Both are described through a logic formalism, but with different approaches. *Joint Intentions* focus on a team's joint mental state, while *Shared Plans* focus on the agent's intentions towards its collaborator's action or towards a group's joint action.

Tools as *STEAM* [30] and *Machinetta* [26] had good results in many complex environments [7] [31] [15] [23] [20] [2] [25] [33]. The natural way would be to use some of these tools to improve teamwork on bot's development in computer games. However, practical issues have pointed out for the development of a new specialized tool. *STEAM* was developed using an old version of *SOAR* [13], that made changes in its language since then. *Machinetta*, that is a successor of *STEAM*, has shown many limitations for the game domain.

In order to overcome these limitations, this work introduces *TWProxy*, a new tool that enables agents to perform teamwork in highly dynamic environments. *TWProxy* uses many of the good ideas of *Machinetta*, avoiding some of its limitations and adding new features.

The remainder of the paper is organized as follows: Section 2 shows the main related works, Section 3 describes the features of *TWProxy*, Section 4 presents experiments and evaluation, and Section 5 concludes the paper.

2 Related Works and Main Contributions

Theoretical works in agent teamwork [14] [5] [6] define some features about team behavior. They describe what the agents need to share in order to perform a teamwork, like goals to achieve, plans to perform together, and knowledge about the environment. The agents need to share their intentions to perform a plan in order to achieve a common goal, the teammates need to be aware about their capabilities and how to fill roles to perform the high level team plan. They must be able to monitor both, their own progress and the group's progress to achieve the team goals. Some systems have been developed using the teamwork idea as teams with human collaboration [23], teams for disaster situation [19], and teams for industry production line [8].

Joint Intentions [14] focus on teammates' joint mental states. A team jointly intends a team action if team members are jointly committed to completing such team action, while mutually believing that they were doing it [30]. *Joint Intentions* assume that the agents are modeled in a dynamic multi-agent environment, without complete or correct beliefs about the world or other agents. They have changeable goals and their actions may fail. Thus, the teammate jointly commits to make public the knowledge when a goal is achieved, impossible to be reached, or irrelevant.

In contrast with *Joint Intentions*, the concept of *SharedPlans* [5] relies on a novel intentional attitude, *intending-that*. An individual agent's *intending-that* is directed toward its collaborator's action or toward a group's joint action. It is defined through a set of axioms that guide an individual to take actions, including communication actions, that either enable or facilitate its teammates to perform the assigned tasks [30].

Based on these theories, important teamwork infrastructures have been developed, like *GRATE** [8], *COLLAGEN* [21], *STEAM* [30], TEAMCORE [20] *RETSINA* [28], *MONAD* [32], and *Machinetta* [26]. *STEAM* was developed to provide flexibility and re-usability of agent coordination through the use of a general model of teamwork. Such model exploits the autonomous reasoning about coordination and communication. Previous works have failed either in fulfilling responsibilities or in discovering unexpected opportunities, once they have used precomputed coordination plans, which are inflexible. Indeed, they have been developed for a specific domain, what hampers the reuse. *STEAM* is specially based on *Joint Intentions* in order to make the basic building blocks of teamwork, and it uses *Shared Plans* to avoid an agent to undo the actions of another agent.

MONAD defines a multi-agent control architecture based on *STEAM* teamwork model to automatically coordinate the execution of multi-agent tasks. *MONAD* provides the designer with tools to specify different protocols and the situations in which they should be applied. It shows how different teams can be easily configured, leading the teams to differ in performance. In spite of *MONAD* be applied to Unreal Tournament[3], its experiments just evaluate how the team configuration can affect the team performance, but this does not evaluate how efficient a team with this tool can be.

Machinetta is a proxy-based integration infrastructure where there is a symbiotic relationship between proxies and team members. The proxies provide teamwork models reusable for heterogeneous entities. *Machinetta* provides each agent with a proxy and the proxies act together in a team. The *Machinetta* project was build over previous works like *STEAM* and *TEAMCORE proxies* [20], adding features like adjustable autonomy reasoning, a novel role allocation algorithm, and the representation of coordination tasks as roles. The pair agent-proxy execute Team-Oriented Programs(TOP) [24], that are abstract team plans that provide high-level description of the activities to be performed. Such programs specify joint plans of the team, but do not contain all of the required details of coordination and communication. Team plans are reactively instantiated and they allocate roles to capable agents.

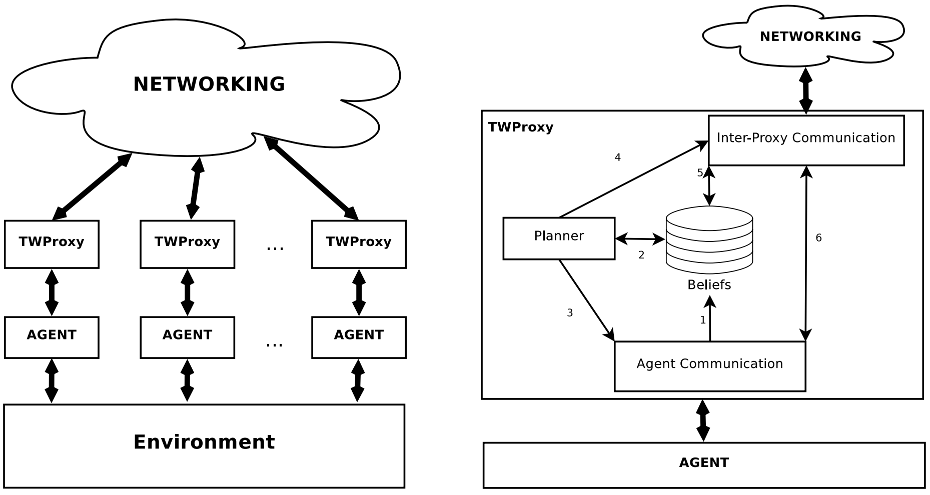
Although there are many tools with similar proposals, none of them focus on the response time for answering the requirements of real-time. In addition, many of these tools show some practical limitation. For instance, *STEAM* was developed using an old version of *SOAR* [13]. *Machinetta* has shown a high response time for highly dynamic domains with real-time requirements. It also does not reuse the terminated plans, being necessary to create extra plans and increasing the memory usage. MONAD is no longer available.

In this paper we present *TWProxy* as a solution to the teamwork problem with real-time requirements. It is based on the *Joint Intentions* formalism and is inspired on *Machinetta*. We add new important features, including: **(i)** a simple and extensible language to describe plans and beliefs, **(ii)** maintenance of consistency through atomic communication, **(iii)** reuse of terminated plans, and **(iv)** plans that are invariant to the number of agents.

[3] A computer game that provides highly dynamic environment.

3 TWProxy

In order to meet the requirements for the development of teams of agents in highly dynamic environments with real-time requirements[4], we introduce *TWProxy*, a new lightweight and efficient infrastructure to provide coordination to teams of heterogeneous agents.



(a) Teamwork model based on proxy (b) TWProxy organization

Fig. 1. TWProxy model

In our approach, summarized in Figure 1(a), each agent is associated with a proxy that performs the communication and the multi-agent planning. The team coordination is achieved using role allocation. Each proxy knows the capabilities of the other teammates. When plan preconditions are achieved, roles are assigned according to this plan. When the plan postconditions are achieved, each teammate releases its role. The proxy does not tell how to execute the role, it just deliberates about what to do. Because of the high-level coordination, a flexible teamwork is possible, leaving the details about role execution with the agent.

Figure 1(b) shows the *TWProxy* internal organization, composed by planner, beliefs base, agent communication interface, inter-proxy communication module and interaction between these elements. The communication modules are easily extensible in order to achieve new organization requeriments or the *TWProxy* integration with new kind of agents. The planner and set of beliefs are driven by the description language of plans and beliefs, as shown in listings 1. Follow are described the *TWProxy* internal interaction represented by arrows in Figure 1(b):

[4] In this work, we assume a response time window of 200 ms.

1. New belief or an update of beliefs, that was perceived by agent and that will update the set of beliefs.
2. Usage of the set of beliefs by planner, that uses stored beliefs to check condition to activate or to deactivate plans.
3. Deliberation about role allocation to the associated agent.
4. Deliberate about role allocation to other teammate or request of capability update, been sent to the responsible proxy, that will request about its associated agent capability update.
5. New belief from other proxy.
6. Channel used to agents communication through *TWProxy*.

3.1 Plans and Beliefs

The initial agent beliefs and its plans are stored in a structured file specified with the language shown in listing 1. This file contains agent beliefs about capabilities, team composition and the specific domain. It also describes high-level team plans.

Listing 1. Grammar of the language to define the plans and initial beliefs

```
          <blf> ::= <block> | <block> <blf>
        <block> ::= "belief" <belief_block> |
                    "plan" <plan_block>
 <belief_block> ::= <id> "{" <inside_belief> "}"
<inside_belief> ::= <attr_value> |
                    <attr_value> <inside_belief>
   <attr_value> ::= <id> ":" <value> ";"
        <value> ::= <id> | <id> "," <values> |
                    <number> | <number> "," <value> |
                    "true" | "false"
   <plan_block> ::= <id> "{" <inside_plan> "}"
  <inside_plan> ::= "roles" ":" <role_values> ";"
                    <pre_block> <post_block>
  <role_values> ::= <id> | <id> "," <role_values> |
                    <id> "+" | <id> "+" <role_values>
    <pre_block> ::= "precondition" "{" <expr> "}"
   <post_block> ::= "postcondition" "{" <expr> "}"
         <expr> ::= "(" <expr> ")" |
                     <id> "." <id> <comp> <number> |
                     <id> "." <id> <comp> <id> "." <id> |
                     <id> "." <id> <comp> "true" |
                     <id> "." <id> <comp> "false" |
                     <expr> "|" <expr> | <expr> "&" <expr>
           <id> ::= "[a-zA-Z][a-zA-Z_0-9]*"
       <number> ::= "[+-]?[0-9]+|[+-]?[0-9]+\.[0-9]+"
         <comp> ::= "==" | "!=" | "<" | "<=" | ">" | ">="
```

Teams of agent-proxy pairs execute Team-Oriented Programs (TOPs) [24]. These TOPs specify the joint plans of the team and their inter-dependencies, but they do not contain all the required details of coordination and communication.

Listing 2. An example about capability belief

```
belief capability_agent001_CTFProtectTheBase {
    type: capability;
    rapId: agent001;
    roleId: CTFProtectTheBase;
    ability: 76;
}
```

Listing 2,3 and 4 show examples of different parts of the beliefs file for the *Capture The Flag* domain. Listing 2 defines that the agent *agent001* can perform the role *CTFProtectTheBase* and that his level of ability for this role is 76.

Listing 3. An example about domain specific belief

```
belief flag_enemy{
    type: flag;
    owner: enemy;
    have: false;
}
belief flag_friend{
    type: flag;
    owner: friend;
    have: true;
}
```

The domain specific belief is also described in the beliefs file, as shown in listing 3. The belief contents is flexible enough to describe new properties, because the new attributes are handled dynamically.

Listing 4. A plan example

```
plan p1{
    roles: CTFProtectTheBase, CTFCaptureTheFlag+;
    precondition{
        ( flag_friend.have == true ) &
        ( flag_enemy.have == false )
    }
    postcondition {
        ( flag_friend.have == false ) |
        ( flag_enemy.have == true )
    }
}
```

Listing 4 shows an example of a plan for the *Capture The Flag* context. When a team has its flag and it does not have the enemy flag, plan *p1* is activated launching a role allocation process. This plan specifies two roles to be allocated, *CTFProtectTheBase* and *CTFCaptureTheFlag*, where the second is flexible for more than one allocation, which is indicated by the symbol ”+” at the end of the role’s name. If a team has four agents available, one performs *CTFProtectTheBase*, while the other three execute *CTFCaptureTheFlag*.

3.2 Communication

The module Inter-Proxy Communication uses two kinds of communication, the atomic broadcast [4] and unicast. Each kind of communication is separated in different channels. The atomic broadcast is used to share team beliefs, keeping the distributed blackboard consistent, and unicast is used in the role allocation process.

In order to solve the problem of atomic communication, *TWProxy* has a simple atomic broadcast layer that can be extended to meet requirements of other kinds of organization. This kind of communication is used when a new team belief is available. In other words, when a teammate perceives a new situation that affects the team, he shares it, keeping the knowledge of the team consistent. However, the teammate does not need to share his individual knowledge, decreasing the number of messages exchanged.

The unicast channel is used to perform a simplified form of Contract Net Protocol [27], in order to achieve role allocation. When the team leader perceives that the precondition of a plan was achieved, he asks the other agents about their ability, and after receiving the answers he allocates the roles for the necessary number of agents.

3.3 Planner

The *TWProxy* planner uses team leader to launch a new plan. The team leader may be either statically pre-fixed or dynamically defined at run-time. A simple way to dynamically choose a team leader is giving the leadership position to the first member that requests it, what is easy to do using atomic communication. The use of team leader avoids conflict problems in role allocation, saving time to react to the environment changes. The team leader has updated team beliefs and it can launch a plan consistently, because everyone is committed to share information relevant to the team.

In order to achieve the real-time requirements, the process of role allocation is not thoroughly distributed. After the team leader receives information about the ability of the teammates, it decides by its own about the allocation of the roles. This is the main limitation of *TWProxy*, but it is also the main issue that increases the performance of role allocation. Only a few messages are needed to update the team leader with the team ability, thus the process is optimized to achieve the best performance.

The planner is like a rule-based system. Every plan has rules to both activate it (the preconditions) and to stop it (the postconditions). These rules are checked using the current knowledge. Roles are assigned in the activation of a plan and the involved agents release their roles in the plan stopping.

With *TWProxy* the team leader can use two algorithms to perform the role allocation: an optimal algorithm with complexity $O(n^3)$ or a new heuristic algorithm with complexity $O(n^2 log(n^2))$. The optimal algorithm is the Hungarian or Kuhn-Munkres algorithm, a classical solution to the assignment problem, originally proposed by H. W. Kuhn in 1955 [12] and refined by J. Munkres in 1957 [18]. The heuristic algorithm is the *Normalized Relative Weight (NRW)*, proposed in this work and presented in Algorithm 1.

In the role allocation problem, each agent $a_i \in A$, is defined by its capability to perform roles, being $R = r_1, ..., r_k$ the set of roles. The capability of an agent a_i to perform a role r_j, is quantitatively given by $C(a_i, r_j)$. The matrix M defines the allocation of roles to team members, where $m_{i,j}$ is the value for the ith row and jth column, and $m_{i,j} = 1$ if a_i performs r_j, otherwise $m_{i,j} = 0$.

Thus, the goal in this problem is to maximize the function

$$f(M) = \sum_{a_i \in A} \sum_{r_j \in R} C(a_i, r_j) * m_{i,j}$$

such that

$$\forall i \sum_{1 \leq j \leq k} m_{i,j} = 1$$

Algorithm 1. Normalized Relative Weight

```
Require: $C$ is a subset of all capabilities, and it has just roles that will be assigned.
  for $i = 0$ to $nAgents$ do
    $norma \Leftarrow 0$
    for $j = 0$ to $nRoles$ do
      $norma \Leftarrow norma + C(i * nRoles + j).cap^2$
    end for
    $norma = sqrt(norma)$
    for $j = 1$ to $nRoles$ do
      $C(i * nRoles + j).cap \Leftarrow C(i * nRoles + j).cap/norma$
      $C(i * nRoles + j).agent \Leftarrow i$
      $C(i * nRoles + j).role \Leftarrow j$
    end for
  end for
  $C \Leftarrow sortByCapability(C)$
  for $k = 0$ to $nRoles * nAgents$ do
    $aIdx \Leftarrow C(k).agent$
    $rIdx \Leftarrow C(k).role$
    if $Allocated(rIdx) =$ false and $HasRole(aIdx) =$ false then
      $Allocated(rIdx) \Leftarrow$ true
      $HasRole(aIdx) \Leftarrow$ true
      $Allocation(aIdx) \Leftarrow rIdx$
    end if
  end for
```

The *NRW* algorithm is a modification of the greedy strategy. It considers that each agent has a vector with all the capabilities, and the algorithm uses just a sub-vector of these capabilities related to the roles that should be performed. After that, *NRW* normalizes each subvector from the agent. Thus, each capability may be compared fairly. Then, the algorithm can use a greedy strategy to find a solution.

3.4 Agent Interface

Each individual agent is attached to a *TWProxy* forming a social agent. The communication between the individual agent and the *TWProxy* is defined by a flexible interface, allowing interaction with several kinds of agents.

The interface between proxy and agent is defined by a base class that must be extended to implement this communication. Such interface is based on message exchange and it can be adjusted to each kind of agent. In a new domain, the interface with the agent is the main modification needed. Thus, it is possible to build a team, to a new domain, by just developing the interface between agent and proxy, and the team program. The individual agent does not need to have social commitment, because *TWProxy* does this for it.

4 Experiments and Evaluation

In this section we present three kinds of experiments to show the main contributions of *TWProxy*. The first one evaluates the role allocation process, comparing the Hungarian algorithm with the heuristic proposed here. The second measures the period of teamwork inconsistency, evaluating the elapsed time between an important change in the environment and the team adoption of a new strategy. The third kind of experiment evaluates the efficiency of bots using *TWProxy* in matches.

4.1 Role Allocation Efficiency

The first experiment about the efficiency of the role allocation intends to explain why to use an heuristic algorithm if there exists an optimal one that runs in polynomial time. The answer is simple, the heuristic algorithm scales better than the optimal one. Considering one second as the maximum time to wait for role allocation, the optimal algorithm scaled up around 230 agents while the heuristic scaled up around 1200 agents. For this reason, *TWProxy* uses by default the Hungarian algorithm when the number of agents is lower than a given threshold, otherwise the proxy uses the heuristic method.

An important issue when dealing with heuristic algorithms is to know how good the heuristic is. A statistical analysis sounds a good solution. Thus, many role allocations were simulated to evaluate the difference between the optimal and the heuristic algorithm. The result of 999 allocations with the number of agents ranging from 3 to 100 is shown in Figure 2 representing the mean of percentage of optimum allocation and its standard deviation. By increasing the number of agents, the algorithm tends to 98% of the optimum while the standard deviation decreases.

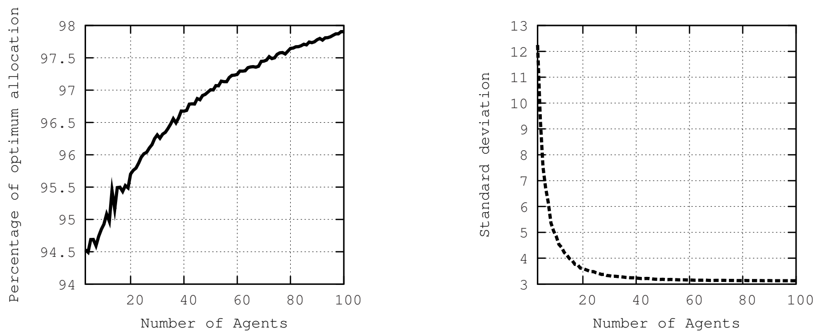


Fig. 2. Means of percentage of the maximum value and standard deviation

4.2 Evaluating the Period of Teamwork Inconsistency

In highly dynamic environments, a great concern in teamwork is to avoid teamwork inconsistency. In the *Capture The Flag* domain[5], a teamwork inconsistency may be the

[5] In the game type *Capture The Flag*, one team must score flag captures by taking the enemy flag from the enemy base and returning it to their own flag. If the flag carrier is killed, the flag drops to the ground for anyone to pick up. If the friend flag is taken, it must be returned (by touching it after it is dropped) before the team can score a flag capture.

following: given a team composed of agents *A*, *B* and *C*, if either *A* or *B* are trying to recover a flag that has been recovered by *C*, then the team is in an inconsistent state. Every team in a dynamic environment will stay in an inconsistent state during some period of time. Therefore, the goal is to minimize inconsistency states.

The experiment about Period of Teamwork Inconsistency(PTI) intended to evaluate the ability of *TWProxy* to react in (soft) real-time to environment changes. In order to compare its performance, *Machinetta* was exposed to the same situation. A group of five agents were updating their beliefs about the flag status, and the time period between the environment change and the role allocation to the whole team has been computed.

The sequence of environment changes was the same for *Machinetta* and *TWProxy*. It was composed of 25 updates, representing a whole match. Each proxy played 100 matches, in order to get a general behavior of PTI. This experiment was performed on a single machine. For this reason, the network latency did not appear in the results.

The results shown in Table 1 describe the mean(μ) and the standard deviation(σ) of the experiments set. The measure unit is milliseconds and the time represents the total time between a new received belief and the modification of the strategy. *TWProxy*, in this experiment, achieved a mean of $77.91ms$, while *Machinetta* got $12303.34ms$. In other words, *TWProxy* was more than one hundred times faster than *Machinetta* to reach a consistent state. The standard deviation shows how stable *TWProxy* execution was in comparison to *Machinetta*.

Table 1. Mean and standard deviation of the PTI experiment

	Elapsed time(ms)	
TWProxy	$\mu = 77.91$	$\sigma = 12.95$
Machinetta	$\mu = 12303.34$	$\sigma = 6459.54$

The performance and stability of *TWProxy* against *Machinetta* was the main result of this experiment. In the midst of this circumstance, *Machinetta* also presented a high usage of memory, more than 100MB per proxy, making its usage difficult in some domains like modern games. *TWProxy* in these experiments did not use more than 1.6MB of memory. The key of these results relies on the kind of application for which *Machinetta* and *TWProxy* were designed. While *TWProxy* design was concerned to achieve real-time requirements, *Machinetta* was originally developed to allow humans to participate in teams with robots and artificial agents. Several points may explain this efficiency, including: the usage of C++ instead of Java; optimization in the access to the beliefs; and mainly no conflicts to solve, because the communication with atomic broadcast guarantees the total order of messages and the usage of team leader avoids the role allocation conflict.

4.3 Teamwork Efficiency

To evaluate the teamwork efficiency of *TWProxy* we used the well known game *Unreal Tournament 2004 (UT2004)*, a multiplayer *First-Person Shooter*(FPS) game. The classical team-based game type *Capture The Flag* was chosen to evaluate the team coherence, because it is easy to measure the results and simple to identify the team behavior. The *GameBOTs* [1] [11] project is used to communicate the developed bot with the game.

The UT2004 game provides an environment with the following characteristics [22]:

- **partially observable** - the agent, using its perception, cannot access all relevant environment states;
- **stochastic** - the next state of the environment is not completely determined by the current state and actions selected by the agents;
- **nonepisodic** - the quality of bot's actions depends on its previous actions;
- **dynamic** - the environment state can change between the agent's perception and agent's actions;
- **continuous** - the number of environment states is unlimited;
- **multi-agent** - there is cooperative and competitive interaction between agents.

In our experiments, the agents have been developed using the framework IAF [17], that implements a hybrid agent architecture [16]. Such agents communicate with the game controlling a player. For these experiments every agent has the same capabilities. Thus, just the teamwork is different. In order to compare *TWProxy* with another teamwork solution, *Machinetta* is used, because of the large number of successful applications in dynamic environments [23] [20] [24] [25].

The team that plays *Capture The Flag* using *TWProxy* has just four plans, that are activated when the state of a flag changes. Such plans generate the following team behavior:

1. when a team has its own flag but it has not the enemy's flag, someone needs to protect the base while others go to capture the enemy's flag;
2. when a team has both, its own and the enemy's flag, all agents must go back to their base;
3. when a team does not have its own flag but it has the enemy's flag, the agent that has the enemy's flag must go back to its base and the others should recover the team's flag;
4. when the team has no flags, every agent must search any flag.

The matches experiments were composed of 30 matches with 15 minutes and teams were composed of 5 players (except in experiments against humans, when 4 players were used). The following teams participated in the matching experiments: TWproxy-Team, a team composed of agents developed with *IAF* and using *TWProxy*; UTBots-Average, the *Capture The Flag* team from UT2004 in *Average* difficulty level (default level); UTBots-Experienced, the *Capture The Flag* team from UT2004, in *Experienced* difficulty level; Machineta-Team, a team composed of agents developed with *IAF* and using *Machineta*; and Human-Team, a team composed of humans with different levels of skill.

The Tables 2, 3, 4, 5 and 6 summarize the results of matches, showing mean (μ) and standard deviation (σ) of the team score and the sum of individual scores. The team score represents how many flags were successfully captured and the sum of individual scores indicates the teammate behavior, computed by the number of killed enemies and captured flags.

TWProxy-Team vs UTBots-Average. *TWProxy-Team* had no difficulties to win *UTBots-Average*, winning 93.33% of the battles, as seen in Table 2. Being *UTBots-Average* a centralized approach that accesses the full state of the game to decide what to do, the result of this experiment represents the possibility to use distributed artificial intelligence in domains like modern games. The immediate advantage of the distributed approach is the possibility of new gameplay, despite increasing the complexity to develop bots.

Table 2. *TWProxy-Team* versus *UTBots-Average*

	TWProxy-Team		**UTBots-Average**	
Wins	93.33%		0%	
Team Score	$\mu = 2.42$	$\sigma = 1.45$	$\mu = 0.03$	$\sigma = 0.18$
Sum of Individual Score	$\mu = 230.97$	$\sigma = 31.11$	$\mu = 100.77$	$\sigma = 20.27$

TWProxy-Team vs UTBots-Experienced. In this experiment, *TWProxy-Team* also had no difficulties to win *UTBots-Experienced*, as shown in Table 3. Although the score of *UTBots-Experienced* was better than *UTBots-Average*, in comparison with the previous experiment, *TWProxy-Team* won 76.67% of the battles.

Table 3. *TWProxy-Team* versus *UTBots-Experienced*

	TWProxy-Team		**UTBots-Experienced**	
Wins	76.67%		0%	
Team Score	$\mu = 1.37$	$\sigma = 1.24$	$\mu = 0$	$\sigma = 0$
Sum of Individual Score	$\mu = 227.03$	$\sigma = 27.32$	$\mu = 117.87$	$\sigma = 27.72$

Machinetta-Team vs UTBots-Average. Using the same agents of *TWProxy-Team*, but with a different proxy (*Machinetta* instead of *TWProxy*), the performance of *Machinetta-Team* was worse than the performance of *TWProxy-Team* in the same situation. Table 4 shows the results of these experiments, where *Machinetta-Team* won only 20% of the battles. Thus, it is possible to see how a tool of coordination can influence the team performance.

Table 4. *Machinetta-Team* versus *UTBots-Average*

	Machinetta-Team		**UTBots-Average**	
Wins	20%		0%	
Team Score	$\mu = 0.27$	$\sigma = 0.64$	$\mu = 0.03$	$\sigma = 0.18$
Sum of Individual Score	$\mu = 185.47$	$\sigma = 48.28$	$\mu = 73.87$	$\sigma = 31.84$

TWProxy-Team vs Machinetta-Team. In these experiments, as shown in Table 5, *Machinetta-Team* did not score on *TWProxy-Team*, while *TWProxy-Team* scored on average 0.6 points per match. In general, *TWProxy-Team* won 43.33% of the battles, while *Machinetta-Team* did not win any one. Although the agents were the same in both teams, *TWProxy* shows advantages over *Machinetta* for this domain, that is a highly dynamic environment with real-time requirements.

Table 5. *TWProxy-Team* versus *Machinetta-Team*

	TWProxy-Team		**Machinetta-Team**	
Wins	43.33%		0%	
Team Score	$\mu = 0.6$	$\sigma = 0.81$	$\mu = 0$	$\sigma = 0$
Sum of Individual Score	$\mu = 217.27$	$\sigma = 57.15$	$\mu = 176.07$	$\sigma = 29.61$

TWProxy-Team vs Human-Team. This last experiment aims to evaluate the team coherence, putting *TWProxy-Team* against *Human-Team*. Usually, teams of humans have the ability to explore some coherence fault in a team of bots, what justifies this experiment. Thus, a group of 25 people, with different skills in FPS[6], participated in the team of humans. Table 6 shows how disputed these battles were. *TWProxy-Team* won 20% of the matches and *Human-Team* won 26.67%. The individual score indicates that the human players were better than bots, however, no incoherence was found in the team of bots. The main advantage for *Human-Team* was the evolving of strategy, because the strategy of *TWProxy-Team* did not evolve. Nevertheless, the results were so close, with a little advantage for *Human-Team*.

Table 6. *TWProxy-Team* versus *Human-Team*

	TWProxy-Team		**Human-Team**	
Wins	20%		26.67%	
Team Score	$\mu = 0.23$	$\sigma = 0.43$	$\mu = 0.5$	$\sigma = 0.97$
Sum of Individual Score	$\mu = 108.73$	$\sigma = 25.02$	$\mu = 154.2$	$\sigma = 39.23$

5 Conclusions

This work introduces *TWProxy*, a new tool based on a team-proxy approach to coordinate groups of agents. Experiments presented here have shown that it is possible to use the teamwork paradigm for highly dynamic environments with real-time requirements. These also confirms the flexibility of high-level planning and the reusability of the proxy approach.

TWProxy shows new important features to develop teams in complex environments. It provides two partially distributed algorithms to perform role allocation, including a new efficient heuristic algorithm. It also provides a simple and extensible language to

[6] First Person Shooter.

describe plans and beliefs. The plans can be reusable and the number of agents performing a plan can be variable. A distributed blackboard with team beliefs is available to the agents, and the use of *TWProxy* for new domains just requires to write the agent's interface and the TOP.

Based on experiments, *TWProxy* has shown advantages compared to *Machinetta*, being more stable and efficient. These experiments evaluated the following aspects: the quality of the role allocation performed by *TWProxy*; the period of teamwork inconsistency, where *TWProxy* has much better response time than *Machinetta* in a simulated environment; and the efficienty of teamwork, where it was possible to see *TWProxy* keeping the team coherence in matches against *UTBots*, *Machinetta-Team* and *Human-Team*. The comparison between *TWProxy* and *Machinetta* was also important to show how the teamwork efficiency may change according to the used coordination tool.

As future works, we intend to implement support to new organization of agents. We also intend to develop a new layer of fault-tolerant communication, trying to minimize the delay inserted in the treatment of lost messages. New games beyond the *Capture The Flag* will also be investigated.

References

1. Andrew Scholer, G.K.: Gamebots (2000), `http://www.planetunreal.com/gamebots` (last access on September 2008)
2. Chalupsky, H., Gil, Y., Knoblock, C., Lerman, K., Oh, J., Pynadath, D., Russ, T., Tambe, M.: Electric elves: Applying agent technology to support human organizations. In: Proceedings of IAAI (2001)
3. de Byl, P.B.: Programming Believable Characters for Computer Games. Charles Development Series (2004)
4. Défago, X., Schiper, A., Urbán, P.: Total order broadcast and multicast algorithms: Taxonomy and survey. ACM Comput. Surv. 36(4), 372–421 (2004)
5. Grosz, B.J., Kraus, S.: Collaborative plans for complex group action. Artificial Intelligence 86(2), 269–357 (1996)
6. Grosz, B.J., Kraus, S.: The evolution of sharedplans. In: Foundations and Theories of Rational Agency, pp. 227–262. Kluwer Academic Publishers, Dordrecht (1999)
7. Hill, R., Chen, J., Gratch, J., Rosenbloom, P., Tambe, M.: Intelligent agents for the synthetic battlefield: A company of rotary wing aircraft. In: Innovative Applications of Artificial Intelligence (IAAI 1997) (1997)
8. Jennings, N.R.: Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. Artificial Intelligence 75(2), 195–240 (1995)
9. Jones, R.M., Laird, J.E., Nielsen, P.E., Coulter, K.J., Kenny, P.G., Koss, F.V.: Automated intelligent pilots for combat flight simulation. AI Magazine 20(1), 27–41 (1999)
10. Kaminka, G.A.: The robocup-98 teamwork evaluation session: A preliminary report. In: Veloso, M.M., Pagello, E., Kitano, H. (eds.) RoboCup 1999. LNCS (LNAI), vol. 1856, pp. 345–356. Springer, Heidelberg (2000)
11. Kaminka, G.A., Veloso, M.M., Schaffer, S., Sollitto, C., Adobbati, R., Marshall, A.N., Scholer, A., Tejada, S.: Gamebots: a flexible test bed for multiagent team research. Commun. ACM 45(1), 43–45 (2002)
12. Kuhn, H.W.: The Hungarian method for the assignment problem. Naval Research Logistic Quarterly 2, 83–97 (1955)
13. Laird, J.E., Newell, A., Rosenbloom, P.S.: Soar: an architecture for general intelligence. Artif. Intell. 33(1), 1–64 (1987)

14. Levesque, H.J., Cohen, P.R., Nunes, J.H.T.: On acting together. In: Proc. of AAAI 1990, Boston, MA, pp. 94–99 (1990)
15. Marsella, S., Tambe, M., Adibi, J., Al-Onaizan, Y., Kaminka, G.A., Muslea, I.: Experiences acquired in the design of robocup teams: A comparison of two fielded teams. Autonomous Agents and Multi-Agent Systems 4(1/2), 115–129 (2001)
16. Monteiro, I.M.: Uma arquitetura modular para o desenvolvimento de agentes cognitivos em jogos de primeira e terceira pessoa. In: Anais do IV Workshop Brasileiro de Jogos e Entretenimento Digital, pp. 219–229 (2005)
17. Monteiro, I.M., dos Santos, D.A.: Um framework para o desenvolvimento de agentes cognitivos em jogos de primeira pessoa. In: VI Brazilian Symposium on Computer Games and Digital Entertainment - Computing Track, pp. 107–115 (2007)
18. Munkres, J.: Algorithms for the assignment and transportation problems. Journal of the Society for Industrial and Applied Mathematics 5(1), 32–38 (1957)
19. Nair, R., Ito, T., Tambe, M., Marsella, S.: Robocup-rescue: A proposal and preliminary experiences. In: Proceedings of RoboCup-Rescue Workshop, Fourth International Conference on Multi-Agent Systems (ICMAS 2000) (2000)
20. Pynadath, D.V., Tambe, M.: An automated teamwork infrastructure for heterogeneous software agents and humans. Autonomous Agents and Multi-Agent Systems 7(1-2), 71–100 (2003)
21. Rich, C., Sidner, C.L.: COLLAGEN: When agents collaborate with people. In: Johnson, W.L., Hayes-Roth, B. (eds.) Proceedings of the First International Conference on Autonomous Agents (Agents 1997), pp. 284–291. ACM Press, New York (1997)
22. Russel, S., Norving, P.: Artificial Intelligence - A Modern Approach. Prentice-Hall, Englewood Cliffs (2002)
23. Scerri, P., Pynadath, D., Johnson, L., Schurr, R., Si, M., Tambe, M.: A prototype infrastructure for distributed robot-agent-person teams. In: The Second International Joint Conference on Autonomous Agents and Multiagent Systems (2003)
24. Scerri, P., Pynadath, D., Schurr, N., Farinelli, A., Gandhe, S., Tambe, M.: Team oriented programming and proxy agents: The next generation. In: Proceedings of 1st international workshop on Programming Multiagent Systems (2004)
25. Scerri, P., Pynadath, D., Tambe, M.: Towards adjustable autonomy for the real world. Journal of Artificial Intelligence Research 17 (2003)
26. Schurr, N., Okamoto, S., Maheswaran, R.T., Scerri, P., Tambe, M.: Evolution of a teamwork model. In: Cognition and Multi-Agent Interaction: From Cognitive Modeling to Social Simulation, pp. 307–327. Cambridge University Press, Cambridge (2005)
27. Smith, R.G.: The contract net protocol: High-level communication and control in a distributed problem solver. IEEE Transactions on Computers C-29(12), 1104–1113 (1981)
28. Sycara, K., Paolucci, M., van Velsen, M., Giampapa, J.: The RETSINA MAS infrastructure. Technical Report CMU-RI-TR-01-05, Robotics Institute Technical Report, Carnegie Mellon (2001)
29. Sycara, K., Sukthankar, G.: Literature review of teamwork models. Technical Report CMU-RI-TR-06-50, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA (November 2006)
30. Tambe, M.: Towards flexible teamwork. Journal of Artificial Intelligence Research 7, 83–124 (1997)
31. Tambe, M., Zhang, W.: Towards flexible teamwork in persistent teams: Extended report. Autonomous Agents and Multi-Agent Systems 3(2), 159–183 (2000)
32. Vu, T., Go, J., Kaminka, G., Veloso, M., Browning, B.: Monad: A flexible architecture for multi-agent control. In: Proceedings of the AAMAS 2003, pp. 449–456 (2003)
33. Yen, J., Yin, J., Ioerger, T.R., Miller, M.S., Xu, D., Volz, R.A.: CAST: Collaborative agents for simulating teamwork. In: IJCAI, pp. 1135–1144 (2001)

The MMOG Layer: MMOG Based on MAS

Gustavo Aranda, Carlos Carrascosa, and Vicent Botti

Universidad Politécnica de Valencia
Departamento de Sistemas Informáticos y Computación
Camino de Vera, s/n – 46022 Valencia – Spain
{garanda,carrasco,vbotti}@dsic.upv.es

Abstract. Massively Multiplayer Online Games present a new and exciting domain for service-oriented agent computing as the mechanics of these virtual worlds get more and more complex. Due to the eminently distributed nature of these game systems and their growing necessity of modern AI techniques, it is time to introduce design methods that take advantage of the power of Multi-Agent Systems, Agent Organizations and Electronic Institutions in order to face the challenges of designing a modern Massively Multiplayer Online Game. This article follows previous pieces of work in the line of Multi-Agent Systems and Massively Multiplayer Online Games research into a common ontology to represent the information that agents store and share and towards the use of the THOMAS service architecture and the SPADE agent platform to address these challenges. The main focus of the article is the so called *MMOG Layer*: a software layer which is independent of the environment simulation and the human-interface devices. An example implementation of such layer is also set out.

1 Introduction

Massively Multiplayer Online Games (MMOGs) are an important focus of research, not only because they are economically attractive, but also because a MMOG involves many fields and a large amount of data that is generated by the interactions of many individuals: configuring a MMOG is a relevant source of research. In the field of AI, to model such systems and its dynamics is nowadays a very relevant task[10, 17].

A typical MMOG (such as World of Warcraft™[1] or Age of Conan™[2], two successful games of this kind) gathers a large community of concurrent players in a persistent virtual game world. These players take part in a common game which has no foreseeable end (i.e. the game is never completed or finished) and, along with some synthetic characters and creatures driven by AI, they populate the virtual world and create an online society with its own rules, landscape, economy and idiosyncrasy.

[1] http://www.blizzard.com
[2] http://www.ageofconan.com

MMOGs are similar in nature to established online virtual communities, like Second Life^{TM3} or There^{TM4}, as both have three-dimensional virtual worlds inhabited by the users' online personas and both have a virtual environment in which the activities of the users take place. However, virtual communities usually spot an open policy and an enforcement of freedom among its members. They do not completely lack rules or norms, but in general terms, users are left to do as they please and there are no short or long term goals for the users' personas to accomplish.

On the other hand, a MMOG also presents some degree of freedom to its players, but it is normally built around a set of rules, logics, conventions and other game mechanics that the game system itself enforces onto the players and that, together, represent the actual way of playing that particular MMOG and set what the game is about and how it becomes an enjoyable and challenging experience for the players. Besides, MMOGs have a tradition of integrating the development, evolution and upgrade of the players' *alter egos* into the game mechanics, making it a *de facto* mandatory long-term game goal for most players.

The present paper focuses in presenting one of the different layers of the architecture *MMOGboMAS* (MMOG based on Multi-Agent Systems) detailing, not only the architecture (ontology and agent taxonomy), but also a prototype applied to a concrete game. The layer presented is the **MMOG layer**, where in an abstraction level that is independent from not only the way the user is going to access the game, but also from the virtual environment, the game rules and mechanics are defined and dealed with.

In this way, the ontology defined is a common basic ontology that the agents that form the game can use to express semantic knowledge in common terms. Also this ontology is used as the base foundation for other, more specific, ontologies that can be built to represent knowledge for a given game or game genre. An example of this later kind of ontology is also provided. The language chosen to represent these ontologies is OWL-DL the standard content language for the semantic web, sanctioned by the W3C[5].

On the other hand, the agents pertaining to the *MMOG layer* will allow the creation of virtual organizations and make use of complex services. So, this layer has been designed and implemented according to the specifications of the THOMAS architecture[6][7]. This architecture is an extension of the classic FIPA platform [13] to open systems dealing with dynamic virtual organizations and services ala SOA.

To implement the prototype, the SPADE[11] MAS platform has been used due to its python language-based feature, so that it is very easy to develop quick and functional agent prototypes.

[3] http://www.secondlife.com
[4] http://www.there.com
[5] http://www.w3.org
[6] http://www.fipa.org/docs/THOMASarchitecture.pdf
[7] http://www.dsic.upv.es/users/ia/sma/tools/Thomas/archivos/arquitectura.pdf

There have been other approaches to use agents in online gaming, albeit not exactly in the MMOG space. Most of them are oriented towards achieving better behaviors in Non-Player Characters. Dignum et al. [9] propose a more natural (long-term) behavior of Non-Player Characters through the use of Multi-Agent Systems, and clarify that game design should be adjusted to incorporate the possibilities of agents early on in the process, a statement also fundamental to this line of research. Also, Gemrot et al. [15] take a more practical approach by developing a full framework, called *Pogamut*, to integrate distributed intelligent agents as synthetic opponents and allies (bots) in games powered by the 'Unreal Engine'$^{\text{TM}}$technology. The main objective of the *Pogamut* project is to provide new AI-driven players that can bestow new challenges to the players and learn from their actions, using a distributed AI network that runs outside of the game clients and server. Both approaches take online gaming in general as a domain for agents achieving good results, so it is a natural step forward for agent technology to enter the MMOG space.

In section 2 the concept of *MMOG based on MAS* architecture is provided. In section 3 the *MMOG Layer* is introduced. In section 4 a specific example deriving from the MMOG ontology is presented. Finally, in section 5 some conclusions are presented along with some hints at future works.

2 MMOG Based on MAS Architecture (MMOGboMAS)

This piece of work follows in the footsteps of previous research efforts like [1] and [2] in which games in general and MMOG in particular are researched as natural scenarios for agents and MAS. A MMOG (like most complex systems) can be seen as a system split into several layered subsystems, with each layer being relatively independent and taking care of one aspect of the whole MMOG experience. From the perspective of this work, a MMOG is split into three layers:

– **HCI Layer:** It is the *client-side* of the system, the part of the game running on the players hardware (PC, mobile phone, game console, . . .). It is the user interface that the game provides to the player, i.e. the game *client* software, and it provides the player with a gaming experience (i.e. 3D graphics, sound, . . .). On a computer, for instance, this software does not differ technically from other types of game software (i.e. offline games), and it is normally built around a game engine, giving it agent capabilities. Due to this layered design, the user is able to interact with the same game in different ways according to the device it is using, that is, the *Interface Agent* it interacts with. Its desirable features are efficiency and good usability. This agent is an *external* agent (out of the direct control of the platform) because it is executed in the **client** machine, that is, the players computer. It is a light-weighted agent that provides the user an interface to the MMOG. The InterfaceAgent is capable of triggering a set of actions that output information messages towards the *MMOG Layer* (in fact, to the *ProfileAgent*). A user may have many InterfaceAgents installed: one on a computer, another on a mobile phone, etc. But since all of them represent the same user, all of them end

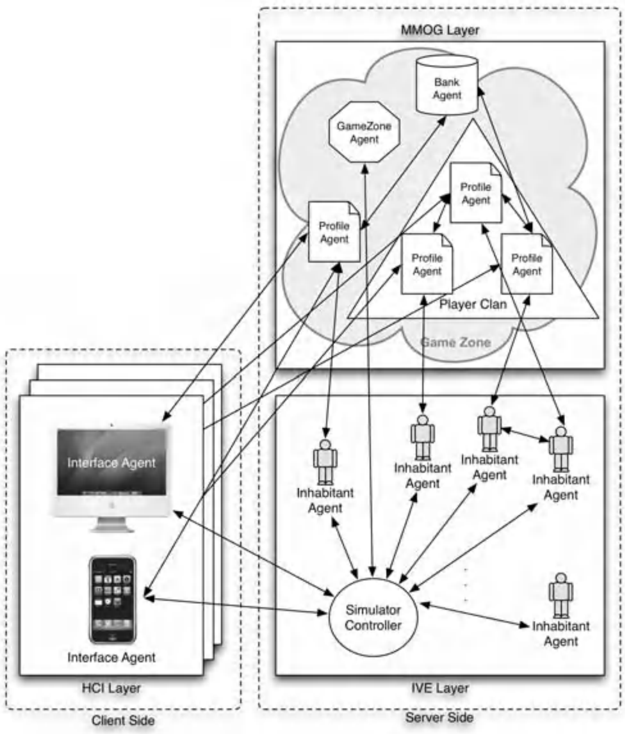


Fig. 1. A MMOG Multi-Agent System

up communicating with the same *ProfileAgent* (representation of the user in the *MMOG Layer*).

- **Intelligent Virtual Environment (IVE) Layer:** It is the virtual representation of the game environment itself. It is part of the *server-side* of the system, the part of the game that runs mostly on the game provider's hardware, a controlled environment. The synthetic *place* and scenario where the game takes place: The virtual world. This world is independent of the type of game or simulation it must give support. The IVE layer falls outside the scope of this article, but it is thoroughly described in [4], presenting two kind of agents: *Simulation Controller Agents* and *Inhabitant Agents*. The first ones are in charge of controlling the environment (or pieces of it) and communicating with the graphical viewers along with the agents inhabiting such environment, that is, the *Inhabitant Agents*.
- **MMOG Layer:** It is a complex subsystem where all the game logics and mechanics are implemented and must be solved at run-time. It is independent of the IVE layer. It implements the game rules / norms controlling the game development. It is the *place* where all the game clients connect to play, and along with the IVE layer must facilitate game server scalability. In this line

of research, this subsystem is seen as the core of a MAS and requires, at least, one agent platform as its foundation.

The main scope of this paper lies within the **MMOG Layer** that will be detailed in the next section.

3 The MMOG Layer

3.1 Agent Taxonomy

As stated before, the **MMOG Layer** is essentially a dedicated, open MAS which runs the game. This MAS uses agent technologies like agent services, Electronic Institutions[6, 12] and Agent Organizations[3, 7, 14] to model some game mechanics, and translates the common issues and situations found in MMOGs into problems that can be solved using classic software agent features, such as agent interactions, agent communication protocols (like auctions or call-for-proposals), service-oriented computing, event-driven behaviors, role models, etc. A (rough) translation table between game and agent concepts can be seen in table 1.

Like any other agentification process, one of the key ideas is to identify the agents and types of agents that will conform the system. In this case, the agents are based on the concepts and entities that form the whole game experience of a MMOG, and are explained in more detail in [2]:

- **ProfileAgent:** a personal agent which manages the player status and profile within the game community. A ProfileAgent is executed server-side, that is, inside the agent platform (which makes it an *internal* agent). It manages the player's preferences in the game world, which avatars the player uses and the **role** that the player plays in the system (*Spectator*, *Player* or *GameMaster*).
- **AvatarAgent:** an agent which represents an avatar of a human player within the game (a PC or Player Character). It is a persistent kind of agent: is not deleted and re-spawned often, it bears a long life cycle. It is the agent that holds the PC *stats* (server-side), and so, a malicious player cannot modify them locally (cheat). The AvatarAgent is the kind of agent that actually performs the actions for the player in the virtual world. It is a persona of

Table 1. Translation between game concepts and MAS concepts

Game Concept	Agent Concept	Technology
Players Clan Players Clan: Lord Players Clan: Objective Players Clan: Loot Share	Agent Organization Agent Organization Supervisor Agent Organization Goal Agent Organization Reward Policy	THOMAS
Game Zone Game Zone Hub	Electronic Institution Federation of Electronic Institutions	AMELI
Client Software Player Character Non-Player Character	Interface Agent *(not in its entirety)* AvatarAgent NPCAvatarAgent	SPADE
Game World	Intelligent Virtual Environment	OSG

the player inside the virtual world and it is a very active type of agent. Since it is an internal agent of the game and it is designed by the game developers, it will strictly follow the game rules, much like a *Governor* agent does not break the rules of an Electronic Institution. It is related to the IVE's Inhabitant Agent (see figure 1).

– **NPCAvatarAgent:** an agent which represents an avatar of an AI-controlled character. It is similar to the AvatarAgent, as both populate the game world, but it does not obey nor represent a player in the game. It is also related to the IVE's Inhabitant Agent.
– **GameZoneAgent:** a kind of agent which implements the logics of the game environment and works as a nexus between the *MMOG Layer* and the IVE Layer's Simulation Controller (see figure 1).
– **BankAgent:** an agent that stores information persistently for other agents and helps maintain the consistency and reliability of the system.

As summarized in table 1, this approach makes use of Agent Organizations and Electronic Institutions too. Agent Organizations are used to express what is known as player Clans or Guilds, i.e. player associations, which may or may not be permanent, and that bond players which have something in common. This commonness is usually a goal or set of goals within the game world, e.g. in a medieval fantasy game like World of Warcraft^{TM} a clan may be formed to build an army to participate in a big-scale battle against opposing armies. In agent terms, these clans are really Agent Organizations within the agent platform. Each organization has its supervisors (which double as *Clan Lords* in the game world), its members (which are AvatarAgents), its goals and its reward policies. The later two are expressed within game terms using an ontology (more on this later in section 3.2).

Electronic Institutions, however, come handy for another task: modeling the game zones. A game zone is a segment of the virtual game world with its own name, theme and uniqueness. For example, in a medieval fantasy game which has a big game world representing an island, a game zone could be a castle called *'Hidden Palace'* and the dungeon below it. Game zones like that work essentially in a very similar way as Electronic Institutions do: there are clear entry and exit points, interconnected rooms, requirements for entering certain rooms, crowd limits, AI-controlled agents that play different roles, etc. In fact, there is a separate branch of the Electronic Institutions called the '3D Electronic Institutions'[6] that cover precisely this kind of virtual places. The whole game world is envisioned as a set of interconnected (or *federated*) Electronic Institutions. The main institution is called the *'Free Zone'*, an Electronic Institution with relaxed constraints and where the Player Clans may be formed.

3.2 A Common Ontology for MMOG Agents

One issue left partially unsolved in previous works, and that this paper addresses, is the need for a common ontology for all these agents to be able to share information in a consistent way (i.e. they need to speak the same content language). It is clear that every game or game genre has its own mechanics, rules and players, but, in essence, all MMOGs share some common concepts, like player avatars. These common concepts can be expressed using a common set of conventions, a common knowledge base. Then, in order to express specific concepts for a given game or genre, custom ontologies that derive from the original one can be described. This common ontology is called **MMOG Ontology**. Besides providing a good starting point for expressing knowledge in MMOGs, the use of a common ontology (or set of ontologies) allows for some interesting developments: first, some parts of the MAS (or even the whole system) can be reused or cloned from game to game with little to no change needed; second, it would be possible to interchange or carry over knowledge between different MMOGs, such as player information, statistics, avatars, etc; third, it settles a common way of working on a MMOG that transcends the current project and helps developer to establish work methodologies; and fourth, it allows for rapid game prototype creation using already available tools for developing ontologies.

The **MMOG Ontology** has been defined using the Web Ontology Language (OWL). More precisely, it is expressed using the OWL-DL (Description Logics) sublanguage for two reasons: one, some of the features of this sublanguage, like defining *one-to-many*-like relational properties, where needed; and two, the full set of the OWL language (called OWL-Full) is not deemed as being computable [5]. Even so, it is a quite light ontology, as it is composed of some base classes that have *datatype*[8] and *object* properties. A summary table of the ontology is presented in table 2. The ontology uses the *'mmog:'* prefix and it presents the following classes: **mmog: Avatar**, **mmog: Clan**, **mmog: GameBeacon**, **mmog: GameItem**, **mmog: GameZone**, **mmog: Goal**, **mmog: Profile**, **mmog: Requirement** and **mmog: RewardPolicy**. Let's review each one of them.

– **mmog:Avatar:** This is the generic class to represent an avatar within the game. It does not matter if its a player-controlled avatar (i.e. played by an AvatarAgent) or an AI-controlled avatar (played by a NPCAvatarAgent). Avatars are uniquely identified by their *hasName* property which ties each of them to a unique name string identifier within the game system.
– **mmog:Clan:** This class represents a Player Clan that the system implements by means of an Agent Organization. It has properties referring its members, lords, goals, reward policies and constraints.
– **mmog:GameBeacon:** A *beacon* is a sort of *'named point'* of the virtual world. A marked position inside a mmog:GameZone which is referred by name. For example, the starting point or one of the exits.
– **mmog:GameItem:** An inanimate object within the game world uniquely identified by a code string. For example, in an adventure game, the items that the players virtually carry in their inventories are instances of a subclass of mmog:GameItem.

[8] A property whose value is a basic data type such integer, float, string, boolean, etc.

Table 2. Summary table of the MMOG ontology

Summary table of mmog:Avatar class

Property	**Type**	**Inverse Property**
atBeacon	*(single mmog:GameBeacon)*	–
hasName	*(single string)*	–
hasOwner	*(single mmog:Profile)*	*owns*
isLordOf	*(multiple mmog:Clan)*	*hasLord*
isMemberOf	*(multiple mmog:Clan)*	*hasMember*

Summary table of mmog:Clan class

Property	**Type**	**Inverse Property**
hasGoal	*(multiple mmog:Goal)*	–
hasLord	*(multiple mmog:Avatar)*	*isLordOf*
hasMember	*(multiple mmog:Avatar)*	*isMemberOf*
hasName	*(single string)*	–
hasRewardPolicy	*(multiple mmog:RewardPolicy)*	–
maxMembers	*(single int)*	–

Summary table of mmog:GameBeacon class

Property	**Type**	**Inverse Property**
hasName	*(single string)*	–
placedAtZone	*(single mmog:GameZone)*	*containsBeacon*

Summary table of mmog:GameItem class

Property	**Type**	**Inverse Property**
hasID	*(single string)*	–
hasName	*(single string)*	–

Summary table of mmog:GameZone class

Property	**Type**	**Inverse Property**
containsBeacon	*(multiple mmog:GameBeacon)*	*placedAtZone*
hasName	*(single string)*	–
hasRequirement	*(multiple mmog:Requirement)*	–

Summary table of mmog:Goal class

Property	**Type**	**Inverse Property**
hasCondition	*(multiple owl:Thing)*	–

Summary table of mmog:Profile class

Property	**Type**	**Inverse Property**
createdWhen	*(single date)*	–
hasContact	*(multiple mmog:Profile)*	*hasContact*
hasPlayerName	*(single string)*	–
owns	*(multiple mmog:Avatar)*	*hasOwner*
playsRole	*(single owl:oneOf {'Spectator' 'Player' 'GameMaster'}*	–

Summary table of mmog:Requirement class

Property	**Type**	**Inverse Property**
atBeacon	*(single mmog:GameBeacon)*	–
hasGoal	*(single mmog:Goal)*	–
playsRole	*(single owl:oneOf {'Spectator' 'Player' 'GameMaster'}*	–
requireClan	*(single boolean)*	–
requireClanMax	*(single int)*	–
requireClanMembers	*(single int)*	–

Summary table of mmog:RewardPolicy class

Property	**Type**	**Inverse Property**
hasAction	*(multiple owl:Thing)*	–

- **mmog:GameZone:** A class to represent a game zone. As stated earlier, a game zone is a segment of the virtual game world with its own name, theme and uniqueness. It holds one or more requirements to be entered.
- **mmog:Goal:** A collection of conditions expressed within valid semantic terms of the game. When all the conditions are met, then the goal is reached. For example, a condition could be the equivalent expression of *'Place object A in beacon XYZ'* or *'Avatar B is a member of Clan C'*.
- **mmog:Profile:** The profile of a player within the game community: The player's screen name along with all the information the system wants to keep from the player, the avatars the player controls, which role the player plays within the game system and the contact list for this player (i.e. the *'friends list'*).
- **mmog:Requirement:** A requirement to enter a given GameZone. It can specify terms regarding player clans attributes, goals or player role.
- **mmog:RewardPolicy:** A reward policy to apply once a goal has been achieved. It holds a collection of actions defined within game terms. Actions will normally be specified using terms from derived ontologies of specific games.

And so, this ontology serves as a foundation to build other, more specific ontologies to express the knowledge of a particular game or type of game. In the next section, an example of these kind of ontologies can be found.

4 Prototyping the MMOG Layer

Analysing the feasibility of the proposal by means of a prototype based on a concrete game, and making use of different agent platforms to allow to manage the different services, virtual organizations and agents involved in this kind of layer.

4.1 Technology Used: THOMAS: Open Systems, Virtual Organizations and Services

As it has been stated before and can be seen in table 1, there are different software technologies used in this implementation of an example of a MMOG layer: to manage Agent Organizations, the THOMAS platform has been used; to implement agents, the SPADE platform has been used for its high prototyping speed; and so on.

The THOMAS abstract architecture is an approach to extend the classic FIPA platform to be used in open systems giving support for dynamic virtual organizations and allowing to express services offered and demanded for the entities in the system *ala* SOA (in fact, every component of the architecture is offering his functionalities as services). Figure 2 shows the THOMAS architecture, which main components are:

- *Platform Kernel*: Component in charge of the communication and internal agent management. It is composed by the *Network Layer* and *AMS* that appears in the classic FIPA architecture.

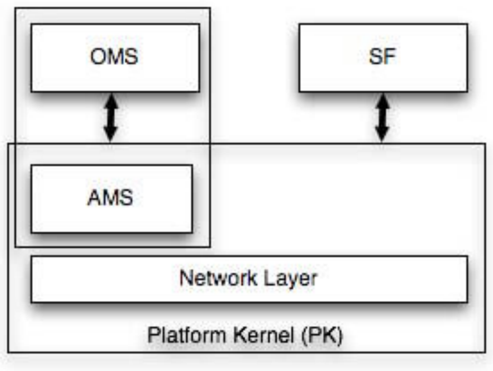


Fig. 2. THOMAS Architecture

- *OMS (Organization Management Service)*: This component controls the life cycle of the virtual organizations in the system.
- *SF (Service Facilitator)*: This is an improvement of the FIPA architecture's DF, so that it allows not only to search and publish services, but also to compose, discover and matchmaking, using SOA standards and techniques.

4.2 An Example: RacingGame

Let's imagine an online racing game where players can assume the roles of drivers of different types of vehicles (Cars, Motorbikes and Trucks) that compete in online races (e.g. a simplified vision of the genre of games like Mini-Racing Online$^{\text{TM}}$[9] or Test Drive Unlimited$^{\text{TM}}$[10]). Players can also participate in an in-game virtual marketplace that allows them to buy and sell parts to upgrade their vehicles. There are basically two kinds of environments within the game: roads (where the racing takes place) and shops (where the selling and buying takes place). Besides, racers can find two types of objects on the road: gas cans (which replenish their gas tanks) and obstacles (which deal damage to a vehicle if it crashes into them). The developers of the game also wish to incorporate some sort of *'Top Racers Ranking'* or *'Leaderboard'*, and so, they need to collect some data from the races, such as how many races has a given player taken part in and how many of them did that player win.

With that game design in mind, a custom ontology called **'RacingGame'** built on top of the *MMOG ontology* can be defined. Let's start by refining the *mmog:GameItem*. Three different kinds of items have been identified: parts, gas cans and obstacles, and such, three distinct subclasses of *mmog:GameItem* can be created to represent them: *VehiclePart*, *GasCan* and *RoadObstacle*. Let's start with the later two. GasCan needs an obvious datatype property called *hasFuel*

[9] http://www.miniracingonline.com
[10] http://www.testdriveunlimited.com

that indicates how many fuel units are contained in the can. It also needs a property to indicate where is it placed. Looking at the *MMOG ontology*, the *placedAtZone* property does just that, and so, it is extended to cover the domain of GasCan too. The RoadObstacle class is very similar, but instead of the *hasFuel* property, it has one called *dealsDamage* which indicates how much damage is dealt to a vehicle if it crashes with the obstacle. The VehiclePart class is going to be the superclass of every subtype of part, so it can have the common properties of every part: *hasPrice*, which is a datatype property indicating how much does the part cost, and *canApplyTo*, a multiple property which indicates that the part can be applied (i.e. is *'compatible'*) to a certain subclass of VehicleAvatar (more on that later). The subclasses of VehiclePart are BodyWork, Engine and Wheels, each one representing one of the main parts of a vehicle and each one having some properties identifying its features and advantages and some other aspects, like the color of the BodyWork (see table 3).

Table 3. Subclasses of VehiclePart

Subclass	Property	Type
BodyWork	hasArmour	*(single owl:oneOf* {1 2 3 4 5})
BodyWork	hasColour	*(single owl:oneOf* {'Red' 'Black' 'White'})
BodyWork	hasWeight	*(single float)*
Engine	gasConsumption	*(single float)*
Engine	hasPower	*(single int)*
Wheels	hasGrip	*(single owl:oneOf* {1 2 3 4 5})

It is clear that the avatars in this game are going to be the vehicles that race, and that all of them have common properties that can be reflected using OWL properties. Let's subclass mmog:Avatar creating the new class **VehicleAvatar** that represents an in-game vehicle. Each vehicle is composed by the union of three parts: a bodywork, an engine and the wheels, and so, the VehicleAvatar class has three object properties called *hasBodywork*, *hasEngine* and *hasWheels* that represent this fact. It also has a numeric property called *maxSpeed* that represents the maximum speed that a vehicle can reach based on the properties of the parts that conform it. VehicleAvatar also has a series of subclasses starting with **BikeAvatar**, **CarAvatar** and **TruckAvatar**, each one representing one type of vehicle that the player can control. These classes refine the type of vehicle, but do not add any significant property (see figure 3).

Regarding zones, two kinds of *mmog:GameZones* are defined: the **CarPartsShop** and the **Road**. The CarPartsShop is the place where player can buy and sell car parts to upgrade their vehicles. It is envisioned to be implemented with an Electronic Institution that resembles those used in other market types (like the one used in the Fishmarket^{TM}system[8]). The other zone is the Road, which represents the virtual courses where the races take place. For this zone, two classes of *mmog:GameBeacon* have been defined to mark the start and end points of a race: **RaceStartPoint** and **RaceEndPoint**.

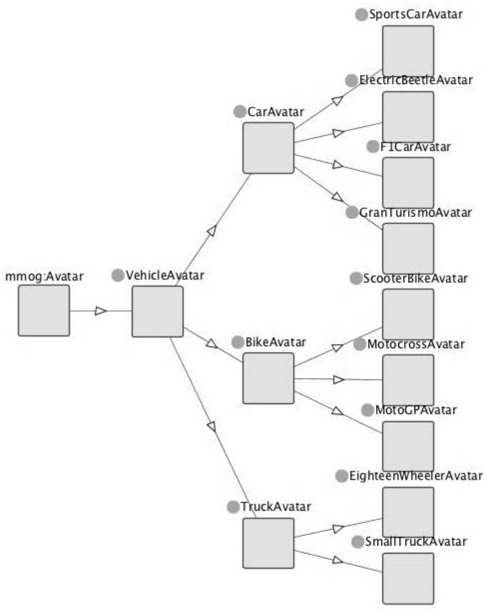


Fig. 3. Diagram of the mmog:Avatar subclasses in the RacingGame ontology

Last, the *mmog:Profile* must be refined in order to represent all the specific data for the races, and such, the **RacerProfile** class is created. It holds an object property called *favoritePlace* which points to the game zone where the player's avatars have spent more time since the moment the player joined the game community (represented by the property *mmog:createdWhen*). It also has a couple of numeric properties to represent the number of races the player has been involved in and how many of them did he actually win: *numberOfRaces* and *numberOfWins*.

4.3 Agents Developed

The SPADE platform has been chosen for it allows prototyping of system agents in a very short amount of time. It can be seen as a *Rapid Application Development* Agent Platform, due in great deal to the use of python as its main programming language and the fact that it is fully FIPA-compliant. This latter compliance with the FIPA standards, allows to easily set up communications between a SPADE agent and THOMAS, as, in its current version, THOMAS is supported by another agent platform supporting FIPA communication.

To illustrate the interplay of these technologies, an example implementation of the Player Clans has been built on top of THOMAS' Agent Organizations. Following the *RacingGame* example, the 'RacingMatches' application facilitates the connection between players interested in becoming part of or creating a brand new race (or clan) and the race management system (Agent Organizations). In this particular example, races are managed inside a THOMAS unit called *Race-Management* which provides search, membership and registration services to the players. This example shows how a VehicleAvatar agent can set up a new race and how other agents can join the race. To set up a race, a VehicleAgent must provide an implementation of one of these pre-defined services: *"BeginnersRace"*, *"AdvancedRace"*, *"DifficultRace"* or *"VeryHardRace"*. It must play the role of *"Organizer"* (or *Provider*) of the race and the other participants play the role of *"Challengers"* (or *Consumers* of the service).

Register in THOMAS: Any VehicleAvatar agent that wants to use THOMAS services, must first register in the THOMAS service platform. This is accomplished by means of sending an *AcquireRole("Virtual", "Member")* message command to the THOMAS OMS. This message implies that the agent wishes to acquire the role 'Member' in the 'Virtual' organization, which is the default unit where all system agents are. Once the OMS checks that the agent can become part of the unit, it adds the agent to the unit (See figure 4).

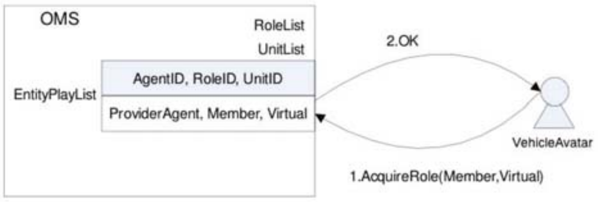


Fig. 4. VehicleAvatar agent joining THOMAS

Register a new service: After registering into a THOMAS platform, an agent can make use of the services offered by the OMS and SF. One of these default services is to look for other services with a similar profile to the one the agent is interested in. In this example, the service is called *BeginnersRace*. So, a VehicleAvatar agent would send the SF a command message like *SearchService(BeginnersRace)*, and the SF then would return the list of registered services with a similar profile and a ranking value for each one of them. This ranking value marks the degree of *affinity* between the given service and the actual result. In the example, the SF may return: *((BeginnersRace,5), (AdvancedRace,4), (DificultRace,3), (VeryHardRace,2))*, which means that there are already similar services (races) in the system and one of them is exactly what the agent is looking for. The VehicleAvatar agent wishes to obtain more semantically detailed information about the *BeginnersRace* service, so it sends a message like *GetProfile ("BeginnersRaceProfile")* to the SF. The SF answers with a URI where the profile is described, and then the VehicleAvatar agent can analyse the profile. This profile, as any semantic document within the THOMAS framework, is written in the OWL-S[16] language, and as such, can be understood by a software agent that can parse and extract information from these kind of documents. The VehicleAvatar can read the service profile and then know that, according to the profile, an agent must adopt a role in order to register a new *process* (a new implementation) of the service. In this example, the *BeginnersRaceProfile* demands the role of *Organizer*, and such, the VehicleAvatar agent can assume that role via a message like: *AcquireRole("Organizer", "BeginnersRace")*.

Implement the new service: Once registered as a provider of a new service, a VehicleAgent must provide a process description (i.e. a valid implementation) of the service it successfully described via the profile. This process description must be accessible through an URI and it is provided by the VehicleAgent to the THOMAS platform with a message like: *RegisterProcess ("BeginnersRaceProfile", "http://.../ BeginnersRaceProcess.owl")*. Normally, the OWL-S documents describing processes are placed in the THOMAS platform, which serves them via a web server (i.e. Apache Tomcat), but SPADE agents can easily serve web documents themselves using platform facilities, and so, a VehicleAgent can host its own process document (see figure 5).

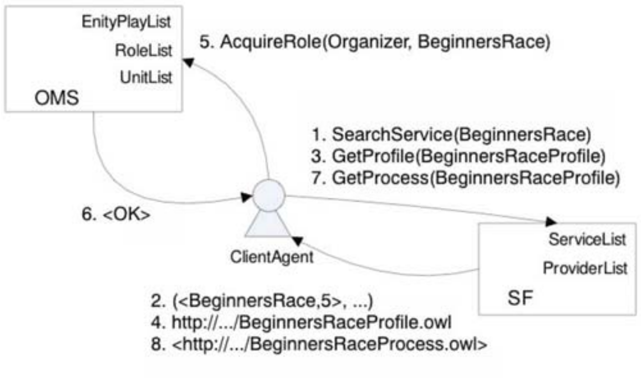

EnityPlayList
RoleList
UnitList
OMS
5. AcquireRole(Organizer, BeginnersRace)
1. SearchService(BeginnersRace)
3. GetProfile(BeginnersRaceProfile)
7. GetProcess(BeginnersRaceProfile)
6. <OK>
ClientAgent
ServiceList
ProviderList
SF
2. (<BeginnersRace,5>, ...)
4. http://.../BeginnersRaceProfile.owl
8. <http://.../BeginnersRaceProcess.owl>

Fig. 5. Register and implement a new service

Client connection: Of course a race is no fun if only the organizer of the event races. So, other VehicleAvatar agents may join the race. To do so, the first step is to join THOMAS as can be seen a few lines above. Then, VehicleAgents can use a *SerchService* kind of message to discover the *"BeginnersRace"* service, and a *GetProfile* message to obtain the profile. After that, they can make use of a message like *GetProcess(BeginnersRaceProfile)* to obtain the process information and know that they must acquire the *Challenger* role through an *AcquireRole("Challenger", "BeginnersRace")* message. And finally, they can make actual use of the service, that is, join an active race, by sending the right command message to the *Organizer* VehicleAgent. In this particular example, such message includes a service call with the following properties as parameters: *VehicleAvatar:hasName* and *Profile:hasPlayerName*.

5 Conclusions and Future Work

This work has continued the research line of **MMOG Multi-Agent Systems** by providing a starting point for building and exchanging semantic content between agents of these kind of systems. The existence of a basic ontology (a common starting point) to represent game-related knowledge is needed in order to achieve greater long-term goals like standardization, inter-systems communication or define a methodology to build these kind of system from the ground up. In this work, such common ontology has been defined using the standard OWL-DL semantic language, as well as a small example of a more refined ontology for a specific game type. Both of them can be downloaded from the web: *http://gti-ia.dsic.upv.es/ontologies/* and are ready for deployment and use.

An interplay between two previously unrelated agents and services platforms, THOMAS and SPADE, has been created in order to allow agents of the **MMOG Layer** to provide and consume services specified in the OWL-S semantic language, which is derived from the OWL language used in the common ontology.

Moreover, the path has been open to properly define and specify the agent interactions and protocols that conform the whole communication scheme of the system, as now there is an ontology for these agents to use and to express their semantic knowledge and an organizational platform to support and enforce services, norms and agent organizations.

Acknowledgements

This work has been partially funded by TIN2006-14630-C03-01, PROMETEO/2008/051 and GVPRE/2008/070 projects and CONSOLIDER-INGENIO 2010 under grant CSD2007-00022.

References

1. Aranda, G., Carrascosa, C., Botti, V.: Intelligent agents in serious games. In: Fifth European Workshop On Multi-Agent Systems (EUMAS 2007). Association Tunnisienne dIntelligence Artificielle (2007)
2. Aranda, G., Carrascosa, C., Botti, V.: Characterizing Massively Multiplayer Online Games as Multi-Agent Systems. In: Corchado, E., Abraham, A., Pedrycz, W. (eds.) HAIS 2008. LNCS (LNAI), vol. 5271, pp. 507–514. Springer, Heidelberg (2008)
3. Argente, E., Palanca, J., Aranda, G., Julian, V., Botti, V., Garcia-Fornes, A., Espinosa, A.: Supporting agent organizations. In: Burkhard, H.-D., Lindemann, G., Verbrugge, R., Varga, L.Z. (eds.) CEEMAS 2007. LNCS (LNAI), vol. 4696, pp. 236–245. Springer, Heidelberg (2007)
4. Barella, A., Carrascosa, C., Botti, V.: Agent architectures for intelligent virtual environments. In: 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, pp. 532–535. IEEE, Los Alamitos (2007)
5. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D., Patel-Schneider, P., Stein, L., et al.: OWL Web Ontology Language Reference. W3C Recommendation 10, 2006–01 (2004)

6. Bogdanovych, A., Berger, H., Simoff, S., Sierra, C.: Narrowing the Gap between Humans and Agents in E-commerce: 3D Electronic Institutions. In: Bauknecht, K., Pröll, B., Werthner, H. (eds.) EC-Web 2005. LNCS, vol. 3590, pp. 128–137. Springer, Heidelberg (2005)
7. Criado, N., Argente, E., Julian, V., Botti, V.: Organizational services for spade agent platform. In: IWPAAMS 2007, vol. 1, pp. 31–40. Universidad de Salamanca (2007)
8. Cuni, G., Esteva, M., Garcia, P., Puertas, E., Sierra, C., Solchaga, T.: MASFIT: Multi-Agent System for FIsh Trading. In: ECAI, vol. 16, p. 710 (2004)
9. Dignum, F., Westra, J., van Doesburg, W.A., Harbers, M.: Games and Agents: Designing Intelligent Gameplay. International Journal of Computer Games Technology, 2009 (2008)
10. Ducheneaut, N., Yee, N., Nickell, E., Moore, R.: "Alone together?": exploring the social dynamics of massively multiplayer online games. In: Proceedings of the SIGCHI conference on Human Factors in computing systems, pp. 407–416 (2006)
11. Escrivà, M., Palanca, J., Aranda, G., García-Fornes, A., Julian, V., Botti, V.: A jabber-based multi-agent system platform. In: Proc. of AAMAS 2006, pp. 1282–1284 (2006)
12. Esteva, M., Rosell, B., Rodriguez-Aguilar, J., Arcos, J.: AMELI: An Agent-Based Middleware for Electronic Institutions. In: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, vol. 1, pp. 236–243 (2004)
13. FIPA. Abstract architecture specification. Technical Report SC00001L (2002)
14. Garcia, E., Argente, E., Giret, A.: Issues for organizational multiagent systems development. In: Sixth International Workshop From Agent Theory to Agent Implementation, AT2AI-6 (2008)
15. Gemrot, J., Kadlec, R., Bida, M., Burkert, O., Pibil, R., Havlicek, J., Zemcak, L., Simlovic, J., Vansa, R., Stolba, M., Brom, C.: Pogamut 3 Can Assist Developers in Building AI for Their Videogame Agents. In: Proceedings of the First International Workshop on Agents for Games and Simulations (2009)
16. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., et al.: OWL-S: Semantic Markup for Web Services. W3C Member Submission 22 (2004)
17. Matskin, M.: Scalable Agent-Based Simulation of Players in Massively Multiplayer Online Games. In: Eighth Scandinavian Conference on Artificial Intelligence: SCAI 2003 (2003)

Architecture for Affective Social Games

Derek J. Sollenberger and Munindar P. Singh

Department of Computer Science,
North Carolina State University,
Raleigh, NC 27695-8206, USA
Tel.: 1-717-860-1809; Fax: 1-919-515-7896
{djsollen,singh}@ncsu.edu

Abstract. The importance of affect in delivering engaging experiences in entertainment and education is well recognized. We introduce the Koko architecture, which describes a service-oriented middleware that reduces the burden of incorporating affect into games and other entertainment applications. Koko provides a representation for affect, thereby enabling developers to concentrate on the functional and creative aspects of their applications. The Koko architecture makes three key contributions: (1) improving developer productivity by creating a reusable and extensible environment; (2) yielding an enhanced user experience by enabling independently developed applications to collaborate and provide a more coherent user experience than currently possible; (3) enabling affective communication in multiplayer and social games.

1 Introduction

Games that incorporate reasoning about their player's affective state are gaining increasing attention. Such games have been prototyped in military training [6] and educational [11] settings. However, current techniques for building affect-aware applications are limited, and the maintenance and use of affect is in essence handcrafted in each application.

We take as our point of departure the results of modeling affect based on appraisal theory. A fundamental concept of appraisal theory is that the environment of an agent is essential to determining the agent's affective state. As such, appraisal theory yields models of affect that are tied to a particular domain with a defined context. Therefore, each new problem domain requires a new affect model instance. A current and common practice has been to copy and edit a previous application (and, occasionally, to build from scratch) to meet the specifications of a new domain. This approach may be reasonable for research proofs-of-concept, but is not suitable for developing production applications.

Additionally, in order to more accurately predict the user's affective state, many affective applications use physical sensors to provide additional information about the user and the user's environment. The number and variety of sensors continues to increase and they are now available via a variety of public services (e.g., weather and time services) and personal commercial devices (e.g., galvanic skin response units). Current approaches require each affective application to interface with these sensors directly. This is not only tedious, but also nontrivial as the application must be adjusted whenever the set of available sensors changes.

To address these challenges we propose a service-oriented architecture, called Koko, that compliments existing gaming engines by enabling the prediction of a gamer's affective state. When compared to existing approaches the benefits of our architecture are an increase in developer productivity, an enhanced user experience, and the enabling of affective social applications. Koko is not another model of emotions but a middleware from which existing (and future) models of affect can operate within. It provides the means for both game developers and affect model designers to construct their respective software independently, while giving them access to new features that were previously impossible. Further, Koko is intended to be used by affective models and applications that seek to recognize emotion in a human user. Whereas it is possible to use Koko to model the emotions of nonplaying characters, many benefits, such as using physical sensors, most naturally apply when human users are involved.

The Koko architecture is realized as a service-oriented middleware which runs independently from the game engine. The primary reason for this separation becomes more apparent when we discuss the social and multiplayer aspects of Koko. Such an approach is consistent with existing techniques for multiplayer access to a central game server.

1.1 Contributions

Any software architecture is motivated by improvements in features such as modularity and maintainability: you can typically achieve the same functionality through more complex or less elegant means [16]. Of course, an improved architecture facilitates accessing new functionality: in our case, the sharing of affective information and the design of social applications. Koko concentrates on providing three core benefits to affective game developers. In the remainder of this section, we elaborate on these benefits.

Developer Productivity. Koko separates the responsibility of developing an application from that of creating and maintaining the affect model. In Koko, the application logic and the affect model are treated as separate entities. By creating this separation, we can in many cases completely shield each entity from the other and provide standardized interfaces between them.

Additionally, Koko avoids code duplication by identifying and separating modules for accessing affect models and various sensors, and then absorbs those modules into the middleware. For example, by extracting the interfaces for physical sensors into the middleware, Koko enables each sensor to be leveraged through a common interface. Further, since the sensors are independent of the affect models and applications, a sensor that is used by one model or application can be used by any other model or application without the need for additional code.

Quality of User Experience. Abstracting affect models into Koko serendipitously serves another important purpose. It enables an enhanced user experience by providing data to both the application and its affect model that was previously unattainable, resulting in richer applications and more comprehensive models.

With current techniques it is simply not possible for separate applications to share affective information for their common users. This is because each application is independent of and thereby unaware of other applications in use by that user. By contrast, Koko-based applications can share common affective data through Koko. This reduces each application's overhead of initializing the user's affective state for each session, as well as providing insight into a user's affective state that would normally be outside the application's scope.

Such cross-application sharing of a user's affective information improves the value of each application. As a use case, consider a student using both an education and an entertainment application. The education application can proceed with easier or harder questions depending on the user's affective state even if the user's state were to be changed by participation in some unrelated game.

Affective Social Applications. In addition to enabling cross-application sharing of affective data, the Koko architecture enables cross-user sharing of affective data. The concept of cross-user sharing fits naturally into the realm of social and multiplayer games. Through Koko, a user if authorized may view the affective state of other members in their social circle or multiplayer party. Further, that information can potentially be used to better model the inquiring user's affective state.

1.2 Paper Organization

The remainder of this paper is arranged as follows. The background section reviews appraisal theory models. The architecture section then provides detailed description of the components that compose Koko. Finally, the evaluation section demonstrates the merits of the Koko architecture.

2 Background

Smith and Lazarus' [18] cognitive-motivational-emotive model, the baseline for current appraisal models (see Figure 1), conceptualizes emotion in two stages: appraisal and coping. *Appraisal* refers to how an individual interprets or relates to the surrounding physical and social environment. An appraisal occurs whenever an event changes the environment as interpreted by the individual. The appraisal evaluates the change with respect to the individual's goals, resulting in changes to the individual's emotional state as well as physiological responses to the event. *Coping* is the consequent action of the individual to reconcile and maintain the environment based on past tendencies, current emotions, desired emotions, and physiological responses [8].

Koko focuses on a section of the appraisal theory process (denoted by the dashed box), because Koko is intended to model emotions in human subjects. As a result, the other sections of the process are either difficult to model or outside Koko's control. For instance, the coping section of the process is outside Koko's control as it is an action that must be taken by the user.

A situational construal combines the environment (facts about the world) and the internal state of the user (goals and beliefs) and produces the user's perception of the world, which then drives the appraisal and provides an appraisal outcome. This appraisal outcome is made up of multiple facets, but the central facet is "Affect" or current emotions. For practical purposes, "Affect" can be interpreted as a set of discrete states with an associated intensity. For instance, the result of an appraisal could be that you

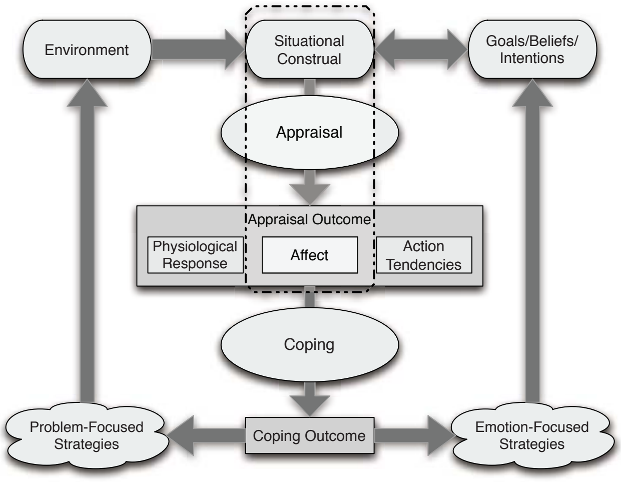


Fig. 1. Appraisal Theory Diagram [18]

are happy with an intensity of α as well as proud with an intensity of β. Unfortunately, selecting the set of states to use within a model is not an easy task as there is no one agreed upon set of states that covers the entire affective space.

Next, we look at three existing appraisal theory approaches for modeling emotion. The first is a classic approach which provides the foundation for the set of affective states used within Koko. The final two are contemporary approaches to modeling emotion with the primary distinction among them being that EMA focuses on modeling emotions of nonplaying characters whereas CARE concentrates on measuring human emotional states.

OCC. Ortony, Clore, and Collins [14] introduced the so-called OCC model, which consists of 22 types of emotions that result from a subject's reaction to an event, action, or object. The OCC model is effectively realized computationally, thus enabling simulations and real-time computations. Further, the OCC's set of emotions have turned out to cover a broad portion of the affective space. Elliot [2] expanded the set of emotions provided by the OCC to a total of 24 emotions. Koko employs this set of 24 emotions as its baseline affective states.

EMA. The EMotion and Adaptation (EMA) model leverages SOAR [12] to extend Smith and Lazarus' model for applications involving nonplaying characters [7]. EMA monitors a character's environment and triggers an appraisal when an event occurs. It then draws a set of conclusions or *appraisal frames*, which reflect the character's perception of the event. EMA then assigns emotions and emotional intensities to each appraisal frame, and passes it to the coping mechanism, which decides the best actions to take based on the character's goals [5].

CARE. The Companion-Assisted Reactive Empathizer (CARE) supports an on-screen character that expresses empathy with a user based on the affective state of the user

[10]. The user's affective state is retrieved in real-time from a pre-configured static affect model, which maps the application's current context to one of six affective states. The character's empathic response is based on these states.

CARE populates its affective model offline. First, a user interacts with the application in a controlled setting where the user's activity is recorded along with periodic responses from the user or a third party about the user's current affective state. Second, the recorded data is compiled into a predictive data structure, such as a decision tree, which is then loaded into the application. Third, using this preconfigured model the application can thus predict the user's affective state during actual usage.

3 Architecture

Existing affective applications tend to be monolithic where the affective model, external sensors, and application logic are all tightly coupled. As in any other software engineering endeavor, monolithic designs yield poor reusability and maintainability. Similar observations have led to other advances in software architecture [16].

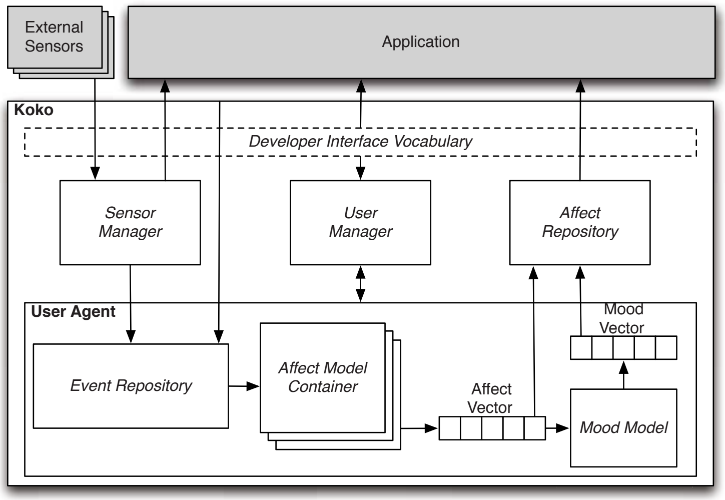


Fig. 2. Koko architectural overview

Figure 2 shows Koko's general architecture using arrows to represent data flow. The following portion of this section provide details on each of Koko's major components. Then after the groundwork of the architecture has been explained we elaborate on the formal interface definitions for the remainder of the section.

3.1 Main Components and Their Usage

Koko hosts an active computational entity or *agent* for each user. In particular, there is one agent per human, who may use multiple Koko-based applications. Each agent has access to global resources such as sensors and messaging but operates autonomously with respect to other agents.

The Affect Model Container. This container manages the affect model(s) for each agent. Each application specifies exactly one affect model, whose instance is then managed by the container. As Figure 3 shows, an application's affect model is specified in terms of the affective states as well as application and sensor events, which are defined in the application's configuration (described below) at runtime.

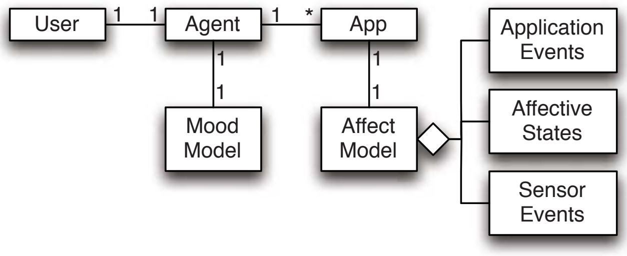


Fig. 3. Main Koko entities

Configuring the affect model at runtime enables us to maintain a domain-independent architecture with domain-dependent affect model instances. Further, in cases such as CARE, we can construct domain-independent models (generic data structures) and provide the domain context at runtime (object instances). This is the approach Koko has taken by constructing a set of generic affect models that it provides to applications. These affect models follow CARE's supervised machine learning approach of modeling affect by populating predictive data structures with affective knowledge. These affect models are built, executed, and maintained by the container using the Weka toolkit [20]. The two standard affect models that Koko provides use Naive Bayes and decision trees as their underlying data structures.

To accommodate models with drastically different representations and mechanisms, Koko encapsulates them via a simple interface. The interface takes input from the user's physical and application environment and produces an *affect vector*. The resulting *affect vector* contains a set of elements, where each element corresponds to an affective state. The affective state is selected from an ontology that is defined and maintained via the developer interface vocabulary. Using this ontology, each application developer selects the emotions to be modeled for their particular application. For each selected emotion, the vector includes a quantitative measurement of the emotion's intensity. The intensity is a real number ranging from 0 (no emotion) to 10 (extreme emotion).

Mood Model. Following EMA, we take an *emotion* as the outcome of one or more specific events and a *mood* as a longer lasting aggregation of the emotions for a specific user. An agent's mood model maintains the user's mood across all applications registered to that user.

Koko's model for mood is simplistic as it takes in affect vectors and produces a *mood vector*, which includes an entry for each emotion that Koko is modeling for that user. Each entry represents the aggregate intensity of the emotion from all affect models associated with that user. Consequently, if Koko is modeling more than one application for a given user, then the user's mood is a cross-application measurement of the user's emotional state.

To ensure that a user's mood is up to date with respect to recent events, we introduce a *mood decay formula* [15] as a way to reduce the contribution of past events. This formula reduces the effect that a given emotion has on the user's mood over time. We further augment our mood model with the concept of *mood intensity* from the work of Dias and Paiva [1]. The mood intensity sums all positive and negative emotions that make up the user's current mood, which is used to determine the degree to which a positive or negative emotion impacts a user. For example, if a user has a positive mood intensity then a slightly negative event may be perceived as neutral, but if the event were to recur the mood intensity would continue to degrade, thereby amplifying the effect of the negative event on the user's mood over time.

Affect Repository. The affect repository is the gateway through which all affective data flows through the system. The repository stores both affect vectors (application specific) and mood vectors (user specific). These vectors are made available to both external applications as well as the affect and mood models of the agents. This does not mean all information is available to a requester, as Koko implements security policies for each user (see user manager). An entity can request information from the repository but the only vectors returned are those they have the permission to access.

The vectors within the repository can be retrieved in one of two ways. The first retrieval method is through a direct synchronous query that is similar to an SQL SELECT statement. The second method enables the requester to register a listener which is notified when data is inserted into the repository that matches the restrictions provided by the listener. This second method allows for entities to have an efficient means of receiving updates without proactively querying and placing an unnecessary burden on the repository.

Event Repository. The event repository is nearly identical to the affect repository with respect to storage, retrieval, and security. Instead of storing vectors of emotion, the event repository stores information about the user's environment. This environmental information is comprised of two parts: information supplied by the application and information supplied by sensors. In either case, the format of the data varies across applications and sensors as no two applications or sensors can be expected to have the same environment. To support such flexibility we characterize the data in discrete units called *events*, which are defined on a per application or sensor basis. Every event belongs to an ontology whose structure is defined in the developer interface.

Sensor Manager. Information about a user's physical state (e.g., heart rate and perspiration) as well as information about the environment (e.g., ambient light, temperature, and noise) can be valuable in estimating the user's emotional state. Obtaining that data is a programming challenge because it involves dealing with a variety of potentially arcane sensors. Accordingly, Koko provides a unified interface for such sensors in the form of the sensor manager. This yields key advantages. First, a single sensor can be made available to more than one application. Second, sensors can be upgraded transparently to the application and affect model. Third, the overhead of adding sensors is amortized over multiple applications.

The sensor manager requires a plugin for each type of sensor. The plugin is responsible for gathering the sensor's output and processing it. During the processing stage the raw output of the sensor is translated into a sensor event whose elements belong to the event ontology in the developer interface vocabulary. This standardized sensor event is then provided as input to the appropriate affect models.

User Manager. The user manager keeps track of the agents within Koko and maintains the system's security policies. The user manager also stores some information provided by the user on behalf of the agent. This includes information such as which other agents have access to this agent's affective data and which aspects of that data they are eligible to see. It is the user manager's responsibility to aggregate that information with the system's security policies and provide the resulting security restrictions to the sensor manager and the affect repository. Privacy policies are crucial for such applications, but their details lie outside the scope of this paper.

Developer Interface Vocabulary. Koko provides a vocabulary through which the application developer interacts with Koko. The vocabulary consists of two ontologies, one for describing affective states and another for describing the environment. The ontologies are encoded in OWL (Web Ontology Language). The ontologies are designed to be expandable to ensure that they can meet the needs of new models and applications.

The *emotion ontology* describes the structure of an affective state and provides a set of affective states that adhere to that structure. Koko's emotion ontology provides by default are the 24 emotional states proposed by Elliot [2]. Those emotions include states such as *joy*, *hope*, *fear*, and *disappointment*.

The *event ontology* can be conceptualized in two parts, event definitions and events. An event definition is used by applications and sensors to inform Koko of the type of data that they will be sending. The event definition is constructed by selecting terms from the ontology that apply to the application, resulting in a potentially unique subset of the original ontology. Using the definition as a template, an application or sensor generates an event that conforms to the definition. This event then represents the state of the application or sensor at a given moment. When the event arrives at the affect model it can then be decomposed using the agreed upon event definition.

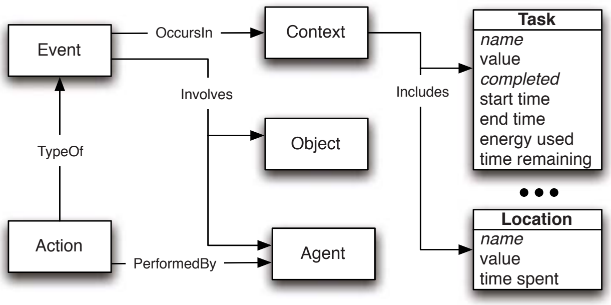


Fig. 4. Event ontology example

Koko comes preloaded with an event ontology (partially shown in Figure 4) that supports common contextual elements such as time, location, and interaction with application objects.

Consider an example of a user seeing a snake. To describe this for Koko you would create an event *seeing*, which involves an object *snake*. The context is often extremely important. For example, the user's emotional response could be quite different depending on whether the location was in a *zoo* or the user's *home*. Therefore, the application developer should identify and describe the appropriate events (including objects) and context (here, the user's location).

3.2 The Architecture Formally

Now that we have laid the groundwork, we describe the Koko architecture in more formal terms from the perspective of an application developer. The description is divided into two segments, with the first describing the runtime interface and the second describing the configuration interface. Our motivation in presenting these interfaces conceptually and formally is to make the architecture *open* in the sense of specifying the interfaces declaratively and leaving the components to be implemented to satisfy those interfaces.

Application Runtime Interface. The application runtime interface is broken into two discrete units, namely, *event processing* and *querying*. Before we look at each unit individually, it is important to note that the contents of the described events and affect vectors are dependent on the application's initial configuration, which is discussed at the end of this section.

Application Event Processing. The express purpose of the application event interface is to provide Koko with information regarding the application's environment. During configuration, a developer defines the application's environment via the event ontology in the developer interface. Using the ontology, snapshots of the application's environment are then encoded as events, which are passed into Koko for processing. The formal description of this interaction is as follows.

$$\mathsf{userID} \times \mathsf{applicationID} \times \mathsf{applicationEvent} \mapsto \perp \tag{1}$$

Upon receipt, Koko stores the event in the agent's event repository, where it is available for retrieval by the appropriate affect model. This data combined with the additional data provided by external sensors provides the affect model with a complete picture of the user's environment.

Application Queries. Applications are able to query for and retrieve two types of vectors from Koko. The first is an application-specific affect vector and the second is a user-specific mood vector, both of which are modeled using the developer interface's emotion ontology. The difference between the two vectors is that the entries in the affect vector are dependent upon the set of emotions chosen by the application when it is configured, whereas the mood vector's entries are an aggregation of all emotions modeled for a particular user.

When the environment changes, via application or sensor events, the principles of appraisal theory dictate that an appraisal be performed and a new affect vector computed. The resulting vector is then stored in the affect repository. The affect repository exposes that vector to an application via two interfaces. Formally, (here a square bracket indicates one or more)

$$\mathsf{userID} \times \mathsf{applicationID} \mapsto \mathsf{affectVector} \tag{2}$$

$$\mathsf{userID} \times \mathsf{applicationID} \times \mathsf{timeRange} \mapsto [\mathsf{affectVector}] \tag{3}$$

Additionally, an application can pass in a contemplated event and retrieve an affect vector based on the current state of the model. The provided event is not stored in the event repository and does not update the state of the model. This enables the application to compare potential events and select the one expected to elicit the best emotional response. The interface is formally defined as follows.

$$\mathsf{userID} \times \mathsf{applicationID} \times \mathsf{predictedEvent} \mapsto \mathsf{affectVector} \tag{4}$$

Mood vectors, unlike affect vectors, aggregate emotions across applications. As such, a user's mood is relevant across all applications. Suppose a user is playing a game that is frustrating them and the game's affect model recognizes this. The user's other affect-enabled applications can benefit from the knowledge that the user is frustrated even if they cannot infer that it is from a particular game. Such mood sharing is natural in Koko because it maintains the user's mood and can supply it to any application. The following formalizes the above mechanism for retrieving the mood vector.

$$\mathsf{userID} \mapsto \mathsf{moodVector} \tag{5}$$

Application Configuration Interface. Properly configuring an application is key because its inputs and outputs are vital to all of the application interfaces within Koko. In order to perform the configuration the developer must gather key pieces of information and then supply that information to Koko using the following interface:

$$[\mathsf{affectiveState}] \times [\mathsf{eventDefinition}] \times [\mathsf{sensorID}] \times \mathsf{modelID} \mapsto \mathsf{applicationID} \tag{6}$$

The *affectiveStates* are the set of states (drawn from the emotion ontology) that the application wishes to model. The *eventDefinitions* describe the structure (created using the event ontology) of all necessary application events. The developer can encode the entire application state using the ontology, but this is often not practical for large applications. Therefore, the developer must select the details about the application's environment that are relevant to the emotions they are attempting to model. For example, the time the user has spent on a current task will most likely effect their emotional status, where as the time until the application needs to garbage collect its data structures is most likely irrelevant. The *sensorIDs* and *modelID* both have trivial explanations. Koko maintains a listing of both the available sensors and affect models, which are accessible by their unique identifiers. The developer must simply select the appropriate sensors and affect model and record their identifiers.

Further, Koko enables affect models to perform online, supervised learning by classifying events via a set of emotions. Applications can query the user directly for the user's emotional state and then subsequently pass that information to Koko. In general, many applications have well-defined places where they can measure the user's responses in a natural manner, thereby enabling online learning of affective state. Applications that do exercise the learning interface benefit from improved accuracy in the affect model. The formal definition of this interface is as follows. Notice there is no output because this interface is used only to update Koko's data structures not to retrieve information.

$$\mathsf{userID} \times \mathsf{applicationEvent} \times \mathsf{emotionClassifier} \mapsto \perp \tag{7}$$

4 Evaluation

Our evaluation mirrors our claimed contributions, namely, the benefits of the Koko architecture. First, we evaluate the contributions of the architecture in itself. Subsequently, we demonstrate the usefulness of the architecture with case studies of both single and multiplayer games.

4.1 Architecture Evaluation

A software architecture is motivated not by functionality but by so-called "ilities" or nonfunctional properties [3]. The properties of interest here—reusability, extensibility, and maintainability—pertain to gains in developer productivity over the existing monolithic approach. In addition, by separating and encapsulating affect models, Koko enables sharing affective data among applications, thereby enhancing user experience. Thus we consider the following criteria.

Reusability. Koko promotes the reuse of affect models and sensors. By abstracting sensors via a standard interface, Koko shields model designers from the details of how to access various sensors and concentrate instead on the output they produce. Likewise, application developers can use any installed affect model.

Maintainability. Koko facilitates maintenance by separating the application from the affect model and sensors. Koko supports upgrading the models and sensors without changes to the components that use them. For example, if a more accurate sensor has been released you could simply unregister the corresponding old sensor and register the new sensor using the old sensor's identifier. Any model using that sensor would begin to receive more accurate data. Likewise, a new affect model may replace an older model in a manner that is transparent to all applications using the original model.

Extensibility. Koko specifies generic interfaces for applications to interact with affect models and sensors as well as providing a set of implemented sensors and models. New sensors and models can be readily installed as long as they respect the specified interfaces.

User Experience. Koko promotes sharing at two levels: *cross-application* or intraagent and *cross-user* or interagent communication. For a social or multi-player application Koko enables users—or rather their agents—to exchange affective states. Koko also provides a basis for applications—even those authored by different developers—to share information about a common user. An application may query for the mood of its user. Thus, when the mood of a user changes due to an application or otherwise, this change becomes accessible to all applications.

4.2 Case Studies

In this section we present two case studies that demonstrate Koko in operation. The first study is the creation of a new social application, called booST, which illustrates the social and multiplayer aspects of Koko. The second study is the re-creation of the affect-enabled Treasure Hunt game, which helps us illustrate the differences between the Koko and CARE architectures.

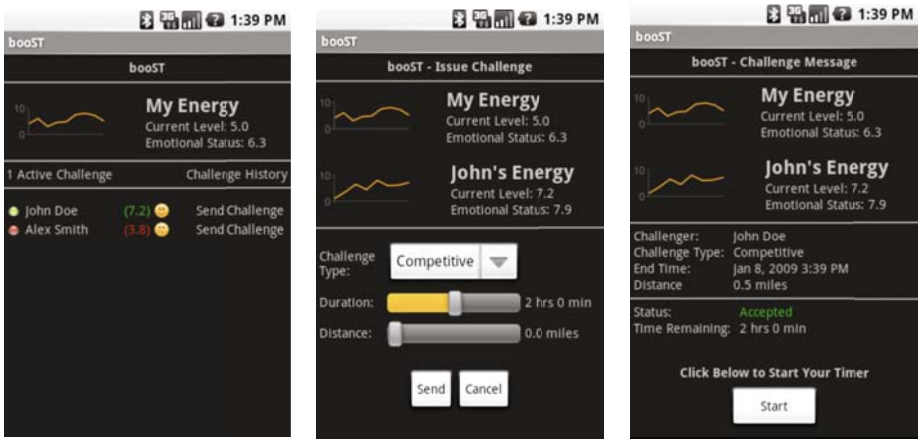
booST. The subject of our first case study is a social, physical health application with affective capabilities, called booST. To operate, booST requires a mobile phone running Google's Android mobile operating system that is equipped with a GPS sensor. Notice, that while booST does take advantage of the fact that the sensor and application run on the same device this is not a restriction that is imposed by Koko.

The purpose of booST is to promote positive physical behavior in young adults by enhancing a social network with affective capabilities and interactive activities. As such, booST utilizes the OpenSocial platform [13] to provide support for typical social functions such as maintaining a profile, managing a social circle, and sending and receiving messages. Where booST departs from traditional social applications is in its use of the *energy levels* and *emotional status* of its users.

Each user is assigned an *energy level* that is computed using simple heuristics from data retrieved from the GPS. Additionally, each user is assigned an *emotional status* generated from the affect vectors retrieved from Koko. The emotional status is represented as a number ranging from 1 to 10. The higher the number the happier the individual is reported to be. A user's energy level and emotional status are made available to both the user and members of the user's social circle.

To promote positive physical behavior, booST supports interactive physical activities among members of a user's social circle. The activities are classified as either competitive or cooperative. Both types of activities use the GPS to determine the user's progress toward achieving the activities goal. The difference between a competitive activity and a cooperative activity is that in a competitive activity the user to first reach the goal is the winner, whereas in a cooperative activity both parties must reach the goal in order for them to win.

Koko's primary function in booST is to maintain the affect model that is used to generate the user's emotional status. The application provides the data about its environment, which in the case of booST is the user's social interactions and the user's participation in the booST activities. Koko passes this information to the appropriate user agent, who processes the data and returns the appropriate emotional status. Further, Koko enables the exchange of affective state with the members of a user's social


1:39 PM
booST
booST
My Energy
Current Level: 5.0
Emotional Status: 6.3
1 Active Challenge | Challenge History
John Doe (7.2) Send Challenge
Alex Smith (3.8) Send Challenge

1:39 PM
booST
booST - Issue Challenge
My Energy
Current Level: 5.0
Emotional Status: 6.3
John's Energy
Current Level: 7.2
Emotional Status: 7.9
Challenge Type: Competitive
Duration: 2 hrs 0 min
Distance: 0.0 miles
Send | Cancel

1:39 PM
booST
booST - Challenge Message
My Energy
Current Level: 5.0
Emotional Status: 6.3
John's Energy
Current Level: 7.2
Emotional Status: 7.9
Challenger: John Doe
Challenge Type: Competitive
End Time: Jan 8, 2009 3:39 PM
Distance 0.5 miles
Status: Accepted
Time Remaining: 2 hrs 0 min
Click Below to Start Your Timer
Start

Fig. 5. booST buddy list and activities screenshots

circle. This interaction can be seen in Figure 5 in the emoticons next to the name of a buddy. The affective data shared among members of a social circle is also used to provide additional information to the affect model. For instance, if all the members of a user's social circle are sad then their state will have an effect on the user's emotional status.

Treasure Hunt. We showed above how a new application, such as booST, can be built using Koko. Now we show how Koko can be used to retrofit an existing affective application, resulting in a more efficient and flexible application.

Treasure Hunt (TH) is a educational game that demonstrates the CARE model [10] wherein the user controls a character to carry out some pedagogical tasks in a virtual world. TH appraises the emotional state of the user based on a combination of (1) application-specific information such as location in the virtual world, user's objective, and time spent on the task and (2) data from physiological sensors providing information on the user's heart rate and skin conductivity. TH conducts an appraisal every time the user's environment changes and produces one of six perceived emotional states is output.

To reconstruct TH using Koko, we first abstract out the sensors. The corresponding sensor code is eliminated from the TH code-base because the sensors are not needed by the application logic. After the sensors have been abstracted, we select the six emotional states from the emotion ontology that match those already used in TH. The final step is to encode the structure of the application's state information into application events. This basic format of the state information is already defined, as the original TH logs the information for its internal affect model.

Both CARE and Koko models can be thought of as having three phases, as outlined in Table 1. The difference between the two architectures is how they choose to perform those phases. For example, in CARE's version of TH the environmental data and emotional classifiers are written to files. The data is then processed offline and the resulting

Table 1. Three phases of CARE affect models

#	Description
1	Gather environmental data and emotional classifiers
2	Perform supervised ML techniques on the data and classifiers
3	Given environmental data produce emotional probabilities

affect model is injected into the application allowing TH to produce the emotional probabilities.

Koko improves on the CARE architecture by eliminating the need for the application to keep record of its environmental data and also by performing the learning online. As a result, the Koko-based TH can move fluidly from phase 1 to phase 3 and back again. For example, if TH is using Koko then it can smoothly transition from providing Koko with learning data, to querying for emotional probabilities, and return to providing learning data. In CARE this sequence of transitions while possible, results in an application restart to inject the new affect model.

This improvement can be compared to an iterative software engineering approach versus the more rigid waterfall approach. CARE corresponds to the waterfall approach, in that it makes the assumption that you have all the information required to complete a stage of the process at the time you enter that stage. If the data in a previous stage changes, you can return to a previous state but at a high cost. Koko follows a more iterative approach by removing the assumption that all the information will be available ahead of time and by ensuring that transitions to a previous state incur no additional cost. As a result, Koko yields a more efficient architecture than CARE.

5 Discussion

This paper shows how software architecture principles can be used to specify a middleware for social affective computing. If affective computing is to have the practical impact that many hope it will, advances in software architecture are crucial. Further, the vocabulary of events and context attributes introduced here can form the basis of a standard approach for building and hosting affective applications.

Game Integration. Due to the social and multiplayer nature of Koko, it cannot be contained within a traditional gaming engine. However, Koko can interoperate with gaming engines in a loosely coupled manner. To incorporate Koko into an existing game engine API, the engine can simply provide a façade (wrapper) around the Koko API. The façade is responsible for maintaining a connection to the Koko service and marshalling or unmarshalling objects from the engine's data structures to those supported by the Koko. Currently, Koko has service endpoints that support the communication of data structures encoded as Java objects, XML documents, or JSON objects.

Affect Modeling. Existing appraisal theory applications are developed in a monolithic manner [2] that tightly couples application and model. As a notable exception, EMA provides a domain-independent framework that separates the model from the application. Whereas EMA focuses on modeling virtual characters in a specific application, Koko models human emotion in a manner that can cross application boundaries.

We adopt appraisal theory due to the growing number of applications developed using that theory. Our approach can also be applied to other theories such as Affective Dimensions [15], whose models have inputs and outputs similar to that of an appraisal model. Likewise, we have adopted Elliot's set of emotions because of its pervasiveness throughout the affective research community. Its selection does not signify that Koko is bound to any particular emotion ontology. Therefore, as the field of affective computing progresses and more well-suited ontologies are developed, they too can be incorporated into the architecture.

Virtual Agents. Koko's interagent communication was developed with a focus on human-to-human social interactions (e.g., booST). This does not limit Koko to only those interactions and we have begun to explore the usage of Koko with human-to-virtual agent interactions. Given the correct permissions, virtual agents (operating outside of Koko) could request the user's affective state in the same manner as agents internal to Koko can. For example, the virtual agents models used in the ORIENT [9] and NonKin Village [17] applications could access the affect state of the player and use that information to enhance the agent's decision making process.

Enhanced Social Networking. Human interactions rely upon social intelligence [4]. Social intelligence keys not only on words written or spoken, but also on emotional cues provided by the sender. Koko provides a means to build social applications that can naturally convey such emotional cues, which existing online social network tools mostly disregard. For example, an advanced version of booST could use affective data to create an avatar of the sender and have that avatar exhibit emotions consistent with the sender's affective state.

Future Work. Koko opens up promising areas for future research. In particular, we would like to further study the challenges of sharing affective data between applications and users. In particular we are interested in exploring the types of communication that can occur between affective agents.

In a companion paper [19], we describe a methodology for building affect-aware, social applications. We are interested in refining and enhancing that methodology for gaming applications that incorporate affect.

References

1. Dias, J., Paiva, A.: Feeling and reasoning: A computational model for emotional characters. In: Bento, C., Cardoso, A., Dias, G. (eds.) EPIA 2005. LNCS (LNAI), vol. 3808, pp. 127–140. Springer, Heidelberg (2005)
2. Elliott, C.: The Affective Reasoner: A Process Model of Emotions in a Multi-agent System. PhD thesis, Northwestern (1992)
3. Filman, R.E., Barrett, S., Lee, D.D., Linden, T.: Inserting ilities by controlling communications. Communications of the ACM 45(1), 116–122 (2002)
4. Goleman, D.: Social Intelligence: The New Science of Human Relationships. Bantam Books, New York (2006)

5. Gratch, J., Mao, W., Marsella, S.: Modeling Social Emotions and Social Attributions. Cambridge University Press, Cambridge (2006)
6. Gratch, J., Marsella, S.: Fight the way you train: The role and limits of emotions in training for combat. Brown Journal of World Affairs X(1), 63–76 (Summer/Fall 2003)
7. Gratch, J., Marsella, S.: A domain-independent framework for modeling emotion. Journal of Cognitive Systems Research 5(4), 269–306 (2004)
8. Lazarus, R.S.: Emotion and Adaptation. Oxford University Press, New York (1991)
9. Lim, M.Y., Dias, J., Aylett, R., Paiva, A.: Intelligent NPCs for Education Role Play Game. In: Dignum, F., Silverman, B., Bradshaw, J., van Doesburg, W. (eds.) Agents for Games and Simulations. LNCS (LNAI), vol. 5920. Springer, Heidelberg (2009)
10. McQuiggan, S., Lee, S., Lester, J.: Predicting user physiological response for interactive environments: An inductive approach. In: Proceedings of the Second Artificial Intelligence and Interactive Digital Entertainment Conference, pp. 60–65. AAAI Press, Menlo Park (2006)
11. McQuiggan, S., Lester, J.: Modeling and evaluating empathy in embodied companion agents. International Journal of Human-Computer Studies 65(4), 348–360 (2007)
12. Newell, A.: Unified Theories of Cognition. Harvard University Press, Harvard (1990)
13. OpenSocial Foundation. Opensocial APIs (2009), `http://www.opensocial.org`
14. Ortony, A., Clore, G.L., Collins, A.: The Cognitive Structure of Emotions. Cambridge University Press, Cambridge (1988)
15. Picard, R.W.: Affective Computing. MIT Press, Cambridge (1997)
16. Shaw, M., Garlan, D.: Software Architecture: Perspectives on an Emerging Discipline. Prentice-Hall, Upper Saddle River (1996)
17. Silverman, B., Chandrasekaran, D., Weyer, N., Pietrocola, D., Might, R., Weaver, R.: NonKin Village: A Training Game for Learning Cultural Terrain and Sustainable Counter-Insurgent Operations. In: Dignum, F., Silverman, B., Bradshaw, J., van Doesburg, W. (eds.) Agents for Games and Simulations. LNCS (LNAI), vol. 5920. Springer, Heidelberg (2009)
18. Smith, C., Lazarus, R.: Emotion and adaptation. In: Pervin, L.A., John, O.P. (eds.) Handbook of Personality: Theory and Research, pp. 609–637. Guilford Press, New York (1990)
19. Sollenberger, D.J., Singh, M.P.: Methodology for engineering affective social applications. In: Proceedings of the Tenth International Workshop on Agent-Oriented Software Engineering. LNCS. Springer, Heidelberg (in press, 2009)
20. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann, San Francisco (2005)

Enhancing Embodied Conversational Agents with Social and Emotional Capabilities

Bart van Straalen, Dirk Heylen, Mariët Theune, and Anton Nijholt

Human Media Interaction, University of Twente,
P.O. Box 217, 7500 AE, Enschede, The Netherlands
(straalenb, d.k.j.heylen, m.theune,a.nijholt)@ewi.utwente.nl

Abstract. In this paper we present our current work on an embodied conversational agent for training medical bad news conversations and discuss the inspiration gained from previous work of our own and others. Central in this research is the influence of emotional and social features on the selection and realization of conversational behavior.

Keywords: Embodied Conversational Agents, Social Agents, Bad News Conversations, Tutoring, Empathy.

1 Introduction

Over the last few decades there has been an important shift in much work on dialogue systems. Traditional spoken dialogue systems were created to fulfill a very specific task, using spoken or written language. By combining spoken dialogue systems with a graphical representation of a human (or human-like entity), so-called Embodied Conversational Agents (ECAs) are able to communicate not only verbally but also nonverbally. In addition, because of their embodiment ECAs are perceived more like intelligent agents endowed with a personality. The work on ECAs thus incorporates more than natural language understanding; it also includes other aspects of cognitive modeling, such as models of emotion and social skills. The last few years have seen a large increase in the research that has been done on modeling emotions [1-5] and on modeling social skills [6-8].

In this paper we present our current work on embodied conversational agents and discuss the inspiration and insight gained from previous work of our own and others. Two types of ECAs are described in order to take stock of part of the state-of-the-art and to point out some of the difficulties the scientific community is dealing with at the moment. We selected tutoring agents and agents in interactive pedagogical drama to discuss, because their behaviors typically contain emotion and social features, which are of great importance in these types of tasks [15]. As the design of our agent, which will hold bad news conversations, also includes these features, we look at these types of agents to gain insight and inspiration about how to implement such features. In addition we look at which developments of traditional types of conversational systems have improved interactions with virtual humans in various ways during the last decade. Components from some of the described systems are taken into account in the design of the agent architecture. Additionally, the design of the virtual agent is based on theoretical models of human cognition to allow virtual humans to behave more as real human beings, both in physical behavior as in mental processes. This approach is founded on the belief that "the best representation of an object is itself." By making virtual humans to be more humanlike, the quality of interacting with such agents will also most likely be improved, opening the door for new improvements and applications.

2 Tutoring Agents

In this section we take a look at virtual tutor systems to see which features play a role in selecting and expressing conversational behavior (in tutoring). In ECA research, tutoring and coaching have been a popular choice of tasks as they display a lot of different aspects of conversational interaction [9-11]. Typically the actions of a tutor includes giving instructions, asking and answering questions, providing explanations, giving examples, setting specific tasks and objectives, motivating the student, providing feedback (both positive and negative) throughout the training session and afterwards, providing support and evaluating the performance of the student. Based on this list the actions of a tutor can be divided into two broad categories: 1) providing information regarding the task at an appropriate level towards the learner and 2) engaging and directing the learner through the learning process. The challenge is to build correct models of the cognitive components that lead to the selection and realization of these actions, as it is often unclear which components lead to a certain behavior, how they function and how they interact with each other.

For current day ECA research the second category is particularly interesting, as it deals with the social and emotional skills that are of great importance when performing the role of tutor. In order to keep a learner motivated and challenged a tutor may need to praise or blame the learner (emotion directed behavior), adopt the role of a study-buddy (altering the social relation between tutor and learner) and keep track of what the learner is thinking and feeling (abducting the mental state of the user). By adapting the virtual tutor's social and emotional skills to each individual learner, the outcome of a tutoring session will most likely be improved significantly. Furthermore it will contribute to the effort to make conversational agent to behave in a more humanlike fashion.

As an example the remainder of this section will describe an intelligent tutoring system we developed called INES (Intelligent Nursing Education Software) [12-14]. The INES system is designed with the purpose of helping students practice nursing tasks using a haptic device within a virtual environment. A virtual human in the INES system provides the role of tutor with which the learner can interact (see figure 1). The virtual tutor is capable of performing acts from both categories mentioned above, but focuses on affective control of the mental state of the learner in the tutoring dialogues [12]. It does so by selecting the appropriate feedback to give to the learner after he or she has performed an action. In order to select the appropriate feedback the tutor makes an assumption about the learner's mental state and consequently adapts the selection of its type of action, the affective language it uses and the overall tutoring strategy. For example the tutor might say "It was quite a difficult task. Try again, but put the needle in more slowly." instead of saying "You put the needle in too fast.

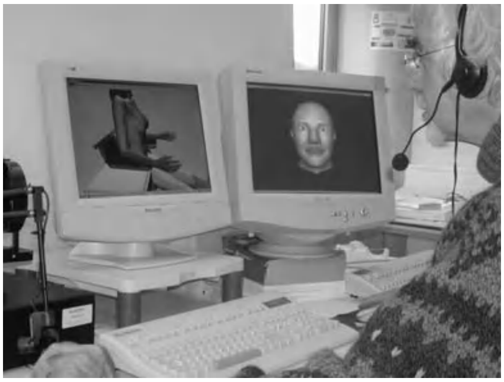


Fig. 1. The INES system. The student is using a haptic device that transforms to an injection needle in the virtual environment (displayed on the *left screen*). In this environment the student can interact with a *Virtual Patient* and perform simple nursing tasks. The performance is monitored by the *Virtual Tutor* (displayed on the *right screen*).

Try again." if the learner comes across as being hesitant. The assumption is based on the observed behaviors of the learner. This includes, amongst others, the learner's confidence level and an appraisal of the learner's actions while he or she is performing the task: Did the learner make many mistakes? How grave were those mistakes? How is the overall performance so far? How (pro-) active is the student? Furthermore, the tutor system also takes into account the difficulty of the task and the emotional effect previous feedback had on the system itself. All these aspects are used to estimate the affective and motivational state of the user (anxious-confident, dispirited-enthusiastic), as well as the performance of the task [15]. As a result the socio-emotional aspects of the interaction between the learner and the virtual tutor do not only influence the learning strategies the tutor adopts but also the manner in which the conversational actions are expressed.

The INES system contains the variables `happy-for' and 'sorry-for' in its mental model that are updated depending on the student's success. The emotions the agent experiences are thus related to the behavior of the learner. The extent to which it responds naturally to the situation is restricted to influencing the learning process in a positive way. These variables are used to adjust the type of feedback.

It is apparent that in general the focus within a tutoring system lies on optimizing the learning achievements of the user, not on making the tutor agent behave as human-like as possible. Although the tutor agent's cognitive capabilities have been improved, e.g. adapting the complexity of the information to the level of the learner and using emotional and social conversational behavior to motivate and engage the learner, these improvements only influence the tutor agent's behaviors to the extent that they increase the performance of the learner. The improved cognitive processes do not cause the tutor agent to act more with its own interests in mind. Nevertheless, the conversational behaviors and the cognitive models of emotions and social skill of tutoring systems described in this section provide us with a good insight on which features play a role in tutoring interactions with humans and how they influence the learner's behavior. By utilizing this knowledge and modifying the cognitive models of emotion and social skills that are described in this section, we aim to create a virtual human that can behave without restricting its behavior by a specific focus.

3 Interactive Pedagogical Agents

This section describes the second type of embodied conversational agents; agents in Interactive Pedagogical Drama [16]. Interactive Pedagogical Drama is a style of educational instruction that has the goal of teaching learners the skills that are necessary to cope with stressful and difficult situations. Within an Interactive Pedagogical Drama, learners interact with believable virtual characters in a story that is recognizable for them and elicits empathy. The goal is that by allowing the virtual characters to face and overcome difficulties, which are similar to those the learners are facing, the learners experience and learn skills that can be used to deal with their own problems. Through interaction with the system the learners can steer the story in such a way that it handles specific problems and solutions they are interested in. While the learner can influence the story, the virtual characters select their actions on their own. Interactive Pedagogical Drama differs from the tutoring systems described in the previous section in the following way. Instead of being actively encouraged by the virtual human to perform the learning task as the virtual tutor instructs, in interactive Pedagogical Drama the learner learns by observing the story. Because the conversational behavior of the virtual characters in interactive pedagogical drama is focused on the story and on other agents instead of on the learner, their behavior selections are less restricted than that of a virtual tutor. Virtual characters in interactive personal drama might also express emotions and social skills that will not surface in a tutoring system, such as anger, frustration or impolite behavior. In order to allow the learners to have a productive interaction with the drama, let them believe in the efficiency of the skills used by the virtual characters and subsequently apply those skills in their own life, it is important that the system has the following characteristics: First of all, the learners must be able to identify themselves with the characters in the story. Secondly the difficulties the virtual characters experience must both be believable and familiar to the learners. If one of these characteristics is lacking the suspension of disbelief of the entire drama fails and the learners will not benefit from the interaction. This means it is vital that the behavior the virtual humans perform is as plausible as possible. Furthermore it can be desirable that when a virtual human is asked why it performs a certain behavior it is able to give a plausible explanation.

A well-known interactive pedagogical drama system that used an agent-based approach is Carmen's Bright IDEAS. Carmen's Bright IDEAS is an interactive health intervention system designed to improve the problem solving skills of mothers of pediatric cancer patients [16]. Parents of children with a chronic disease often lack the capabilities of dealing with many demands their sick child demands and the needs of their spouse, their healthy children and their work. The goal of the system is to teach a specific approach to social decision making and problem solving called Bright IDEAS [17] and to help parents in dealing with difficult situations. The drama of Carmen's Bright IDEAS narrates the following scenario: It relates the problems and stresses of the protagonist of the story, Carmen, who has a nine-year-old son with pediatric leukemia and a six-year-old daughter. Carmen discusses her problems with a counselor,

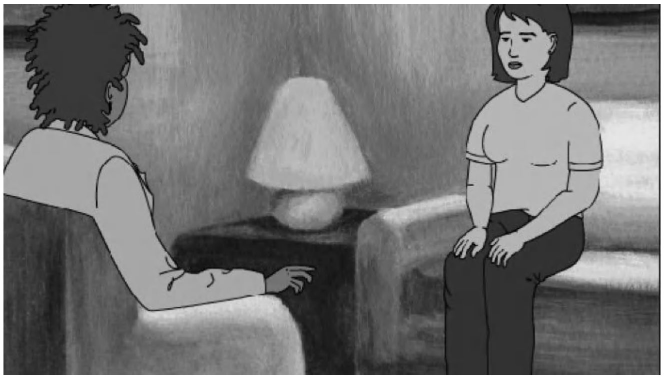


Fig. 2. Carmen's Bright IDEAS. *Carmen*, the agent on the right, discusses her problems with *Gina*.

Gina, who suggest she uses Bright IDEAS to help her to deal with difficult situations. With Gina's help Carmen goes through the initial steps of Bright IDEAS then completes the remaining steps on her own (see figure 2).

In Carmen's Bright Ideas, the user can influence the course of the story at specific points, but does not participate directly as a story character. Instead the learner can control the actions of Carmen at an intentional level by directing the thoughts and feelings Carmen might have in a certain situation when asked about it. Subsequently, the selected thoughts and feelings are incorporated into the mental model of Carmen. This will result in the virtual agent to perform actions which are congruent with the thoughts and feeling that were selected by the learner. The thoughts and emotions the learner can choose from are formalized in such a way that the learner is able to identify them and relate herself to the situation. Where Carmen allows the learner to interact with the drama, Gina's task is to make sure the social problem solving technique is followed. The virtual counselor does so by appropriately responding to the actions of Carmen and motivating her through dialogue and gestures.

The agent architecture in Carmen's Bright IDEAS is based on a multi-layer transition-based agent model called Situation Spaces [18]. This entails that there are specific states in which the agents can find themselves, defined by the situation at that moment. The agents select their behavior based on the state they are in, instead of an event that occurs in the world. For virtual characters in Carmen's Bright IDEAS four different layers exist, each with a variety of states; problem solving, dialogue model, physical focus and emotional appraisal. For the Gina character the problem solving layer is used to give form to the dramatic structure, including the IDEAS steps and the strategies Gina uses to realize these steps. The dialogue model is used to select and execute dialogue acts to bring these strategies about. In Carmen's case the problem solving and dialogue models are more reactive and focus on responding to the communicative actions of Gina. In both characters the physical focus layer is used to manage and execute non-verbal behavior and the emotional appraisal layer covers the agents' emotional appraisal model [16]. Although both virtual characters can make use of non-verbal communicative behavior, it has a larger impact on the learner when it is performed by Carmen.

One of the key points of Interactive Pedagogical Drama is that the learner is able to see the causal relations between a selected intention, the accompanying behavior and the effect that behavior has. As both the selectable intentions and the response behaviors are plausible, the insight in the cognitive processing helps the character to become more human-like. Also, in Interactive Pedagogical Drama the focus of the virtual characters is on the conversation and only secondary on the task of teaching the learner something. In order to facilitate a more appropriate conversation, the behavior selection of the virtual characters will be less restricted than that of tutoring agents where the main focus of their behavior lies on the tutor task. Consequently the virtual characters in an Interactive Pedagogical Drama can select from a much wider range of conversational behaviors and thus will display more diverse emotions and social skills in their dialogues. As our goal is to make an embodied conversational agent that acts like a human, both behavioral and mentally, the insights in the relationship between intentions and behavior and the insights in the emotion and social features gained by research on Interactive Pedagogical Drama are very useful.

4 A Bad News Agent

Having looked at several state-of-the-arts ECA systems, we have gained some insight on which features are important in conversations and how these features influence a (virtual) human's behavior. Whether a conversation with an ECA involves tutoring, counseling, entertaining or is related to another task, it is of great importance that the virtual character's actions are plausible. This can be achieved by trying to aim for realism both externally and internally, by which we mean that the agent performs close to human-like behavior (i.e. physically) and is driven by close to human-like cognitive models. At first glance there are two benefits. First, by modeling human cognitive processes insight is gained in the practical workings of human cognition, albeit at an abstract level and secondly virtual characters will be able to provide a more understandable and realistic explanation of why they have chosen to perform certain behaviors. All in all, this will contribute in making virtual human be more human-like.

In order to realize a realistic virtual human, we are currently designing a virtual character that makes use of cognitive models that are involved in the selection of appropriate conversational behavior i.e. when the virtual human is engaged in an interaction, what kind of behaviors does it select, why does it select them and how do these selection mechanisms influence the manifestation of the behavior? The cognitive models are based on psychological and sociological theories of human cognition. The purpose of the system is to assist physicians (in training) to practice holding bad news conversations. To this end the function of the virtual character is twofold. Primarily it plays the role of the virtual patient that is receiving the bad news and secondly it performs a tutor role, giving feedback to the learner about why it responded the way it did (see figure 3).

The conversational behaviors of the virtual human are not restricted to facilitating the optimal learning experience (e.g. steering the conversation so the learner will be challenged more) as is the case in more traditional tutoring systems, but instead the virtual human is only limited to perform behaviors that are appropriate given the situation.

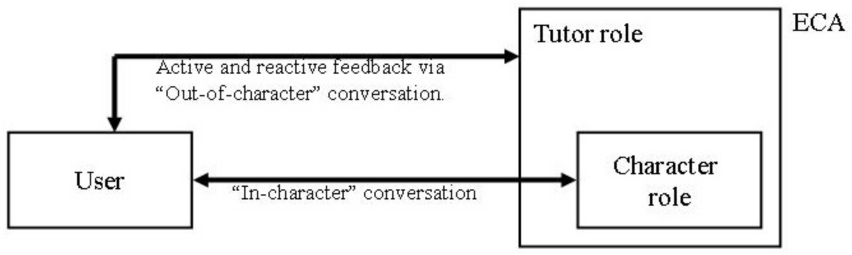

ECA
Tutor role
Active and reactive feedback via "Out-of-character" conversation.
User
"In-character" conversation
Character role

Fig. 3. This is the overall layout of the agent. Notice that the *Character role* is included in the *Tutor role*. This has the advantage that the *Tutor role* can easily refer to the behaviors the *Character role* has performed when the *Tutor role* gives feedback to the *User*.

4.1 Bad News Domain

It is important to understand what we mean by "bad news", so as to place this research in the correct context. In general, a bad news conversation is a dialogue in which the "speaker" discloses information that is unfavorable to the "listener". In this research the topic of the unfavorable information will involve the "listener's / receiver's" medical condition, such as with patients that have a terminal disease. Definitions of bad news as they are used in different studies are: "Any information which adversely and seriously affects an individual's view of his or her future." [19] or "news that will change a patient's outlook for the future in a very negative way. Such bad news can be about a severe illness, prospects of death or increasing levels of limitations." [20]. By using bad news conversations as the task of the interaction with the virtual agent and subsequently trying to model the cognitive processes involved, we hope to gain insight in a variety of advanced cognitive behaviors such as affective and social aspects.

A significant amount of research has been done on how someone should conduct bad news conversations, resulting in several detailed protocols and strategies that describe the best way of delivering bad news [19, 21, 22]. Unfortunately, few of these studies have looked at the response behaviors of the receivers of bad news. We are particularly interested in the way receivers respond, both verbally and non-verbally, to receiving the bad news, why they respond in such a manner and most importantly how such behaviors can be modeled into a virtual human. To this end psychological and sociological literature has been studied on the subject of coping mechanisms and strategies [23, 24]. Of particular relevance is the work of Elizabeth Kübler-Ross, who in her work has gained a great understanding on behavior of terminally ill patients [25]. This has led to the formation of the well-known categorization of coping strategies that are utilized by dying patients: Denial and Isolation, Anger, Bargaining, Depression and Acceptance. By analyzing these different coping mechanisms in terms of affective social aspects we hope to gain an idea on how they influence conversational behavior. So far we have split the influence of coping strategies into two categories: one category that influences the selection of behavior (i.e. which behavior should the virtual agent perform) and another category that influences the manifestation of the selected behavior (i.e. in what manner is the behavior carried out). Contained in the coping strategies are emotional and social features that cause the influence of behavior. A good example is the Anger coping behavior to receiving bad news. When this strategy is adopted the virtual human will select appropriate conversational behavior to cope with the situation. This may be result in selecting the “assigning blame” speech action (attribution emotion) and its accompanying gestures. The selected speech action will be impolite (social relation) and gesturing will have characteristic that are associated with anger (short, strong movements). However in order to use these coping strategies the virtual humans must possess emotions and social skills. To that end we incorporate models derived from cognitive theories.

4.2 Bad News Agent Architecture

The basis of the agent architecture of our Bad News agent is the Beliefs, Desires and Intentions (BDI) cognitive model [26-28], one of the most well-known and most studied models used in creating reasoning intelligent agents [29]. One of the basic components of the architecture is a belief-base that contains all the virtual character’s beliefs about the world (including other agents, such as it human interlocutor, and itself). Beliefs describe the agent’s subjective interpretations of the situation and not an objective representation. For instance, if a patient is given an estimation of one year of remaining life and the normal prognosis is four to six months, he still might believe that one year is short.

For the aspect of desires we intend to include a goal-base that contains a variety of goals the agent may adopt. The main difference between desires and goals is that desires are roughly unrestricted objectives or situations that the agent *would like* to achieve or bring about. Goals on the other hand are those desires that are actively pursued and have to be consistent with each other. For example, an agent might have the desires to *Get out of the hospital* or to *Get treatment in the hospital*. Obviously these are two desires that are not consistent with each other and as such only one can be adopted as a goal.

As described in the BDI model, intentions represent the deliberative state of an agent, i.e. which goal it is committed to and is it trying to achieve by executing its plan of action. However, in our architecture we make a distinction between the terms intention and communicative intent. While the former may be a proper representation of commitment to a goal resulting in a plan of communicative behaviors, the latter is a description of which behavior or state of mind a communicative act is trying to elicit from the interlocutor. For example, if the goal the agent is committed to, (i.e. its intention) is to be comforted by getting a positive reassurance from the doctor, the agent might ask “Everything is going to be okay, isn’t it?”. This question contains two communicative intents. The first intent is to cause the doctor to believe that the agent believes that everything is going to be alright (if he did not think so already). Secondly the question intends to elicit a response in which the doctor confirms the belief of the agent that everything is going to be alright. If the agent forms a communicative intent, this intent will be expressed by its behavior. More specifically, the wished-for effect that a communicative act brings about (i.e. the communicative intent) is contained in the communicative act itself, but it depends on the interpretation by the interlocutor if this effect is achieved. In case of the example, if the doctor ignores or fails to understand the communicative intent he might answer truthfully that everything is NOT going to be alright. A communicative action that does not allow for the possible realization of the communicative intent has no purpose and as such will not be performed. This wished-for effect should not only cause a change in the belief, goal or intention state of the addressee, but should be extended to include the user's affective state. The purpose of the wished-for effect is to get the interlocutor to believe something, know something, do something, feel something etc. that causes the agent to be closer to achieving its own goal.

Although beliefs, desires (or goals) and intentions form a basis for an embodied conversational agent, it is necessary to also incorporate affective and social features in order to create believable interactive virtual humans. To that end we also include a component in the agent architecture that deals with affect. The main content of this component at this point is an emotion appraisal model of our own design, which is based on a conglomeration of features obtained from existing emotion and emotion appraisal models and fitted together. It incorporates features from the OCC model [4], the Affective Reasoner (AR) model [2] and the EMA model [3]. The OCC model is a well-known theoretical model of human emotion. It has been the basis for several state-of-the-art emotion appraisal systems such as EMA and that of FearNot![1]. The OCC model evaluates how the state of the world influences the emotions of the person. However the OCC model does not take into account the mental states of other agents when it is determining which emotion should be elicited based on the situation. In order to make a believable and realistic virtual human this capability needs to be included, so that the virtual agent can generate appropriate responses to the human interlocutor. Therefore we extract features from the Affective Reasoner model that are able to deal with the mental state of others. Also the OCC model does not describe how the formed emotions influence the selection and execution of behavior, which is a problem that needs to be addressed if a virtual human is to be created. Some of the features of the EMA model are included in our affective model as the EMA model utilizes a set of appraisal variables that can be used quite easily in the context of bad news conversations.

In addition to emotion, social skills also play an important role in natural conversation. They influence both the selection of conversational behavior and the realization of that behavior. Each social situation requires a particular type of behaviors. This is represented in the architecture by dividing the virtual human's conversational behavior into categories that correspond with different social situations. Only behaviors from the category that corresponds to the social situation can be selected. Also each category contains a set of labels that are passed on to the behavior realizer to dictate how the behavior should be executed (e.g. volume of speech, specific facial expressions, and characteristics for gestures).

For the input of the cognitive processes we make the assumption that the virtual human has interpreted input signals from the environment. These signals are all the features that make up the conversational behavior of the learner. Verbally this entails, amongst others, the learner's speech act (both the prosody and the content), when the learner speaks and non-verbally this refers to the learner's facial expression, his gaze behavior, his head movement and his body posture as detected by the virtual human. Consequently the virtual human forms a *causal interpretation* [3]. This is a configuration of the agent's belief-base, goal-base and intention at a certain time-point that represents the agent's subjective interpretation of the relationship between the agent and the environment plus the interpreted but not yet fully processed input signals. The causal interpretation is then handled by cognitive processes that are based on cognitive models. These processes include updating the belief-base (including beliefs about the social relationship between the virtual agent and the human interlocutor), updating the goal-base, emotional appraisal, adjusting plan of behaviors and monitoring the environment. As a result of the cognitive processing, an intention (and communicative intentions) is selected. This selection is influenced by the social relation between the virtual human and the environment, the coping strategy the agent has selected and the output of the emotion appraisal. Based on this intention an appropriate category of behaviors is selected from a behavior library. For example, if the intention is to *give an answer to the interlocutor*, then the "answering"-category of behavior is selected. Subsequently, the behavior that is most appropriate, according to the virtual human's current the social state and the emotion appraisal, is selected. In the situation of a bad news conversation it is likely the social relationship with the interlocutor (a doctor) is quite formal. This will lead to a formal, polite type of answer such as "I understand doctor" instead of "yeah okay doc". Additionally the emotional state of the virtual human (most likely dismayed and sad) will result in a terse answer: "I understand doctor" instead of "It is perfectly clear what you are saying to me doctor." The manner in which the behavior is performed (e.g. variables in prosody and gestures) also depends on the emotion state and the social state of the virtual human.

5 General Discussion

Embodied conversational agents can benefit significantly from the inclusion of social and affective features. Although these features can be included in many ways, a good way of creating virtual humans that represent real humans seems to be to incorporate cognitive models of psychological and sociological theories into the design of such agents. Particularly the inclusion of cognitive models that deal with emotions and social skills are important as they greatly increase the capability of virtual human to behave as a real human. Both emotions and social features influence a virtual human's behavior in two manners: first by influencing which conversational behavior the agent selects and secondly by influencing the way this behavior is executed. In our development of the Bad News Agent, we are trying to combine both of these.

Acknowledgements. This research has been supported by the GATE project, funded by the Netherlands Organization for Scientific Research (NWO) and the Netherlands ICT Research and Innovation Authority (ICT Regie).

References

1. Aylett, R., Louchart, S., Dias, J., Paiva, A., Vala, M.: FearNot! - An Experiment in Emergent Narrative. In: Panayiotopoulos, T., Gratch, J., Aylett, R.S., Ballin, D., Olivier, P., Rist, T. (eds.) IVA 2005. LNCS (LNAI), vol. 3661, pp. 305–316. Springer, Heidelberg (2005)
2. Elliott, C.D.: The affective reasoner: a process model of emotions in a multi-agent system. Northwestern University, Evanston (PhD Thesis) (1992)

3. Marsella, S., Gratch, J.: EMA: A computational model of appraisal dynamics. In: Gratch, J., Marsella, S., Petta, P. (eds.) Agent Construction and Emotions, Vienna, pp. 601–606 (2006)
4. Ortony, A., Clore, G.L., Collins, A.: The Cognitive Structure of Emotions. Cambridge University Press, Cambridge (1988)
5. Scherer, K.R.: What are emotions? And how can they be measured? Social Science Information 44(4), 695–726 (2005)
6. Bickmore, T.W.: Relational Agents: Effecting Change through Human-Computer Relationships. MIT, Cambridge (2003); (PhD Thesis)
7. Bickmore, T.W., Schulman, D., Yin, L.: Engagement vs. Deceit: Virtual Humans with Human Autobiographies. In: Ruttkay, Z., et al. (eds.) IVA 2009. LNCS (LNAI), vol. 5773, pp. 6–19. Springer, Heidelberg (2009)
8. Cassell, J., Bickmore, T.W., Billinghurtst, M., Campbell, L., Chang, K., Vilhjálmsson, H., Yan, H.: CHI 1999, pp. 520–527 (1999)
9. Gratch, J., Marsella, S.: Tears and Fears: modeling emotions and emotional behaviors in synthetic agents. In: Agents 2001, pp. 278–285. ACM Press, New York (2001)
10. Johnson, W.L., Rizzo, P., Bosma, W., Kole, S., Ghijsen, M., van Welbergen, H.: Generating socially appropriate tutorial dialog. In: André, E., Dybkjær, L., Minker, W., Heisterkamp, P. (eds.) ADS 2004. LNCS (LNAI), vol. 3068, pp. 254–264. Springer, Heidelberg (2004)
11. Rickel, J., Johnson, W.L.: STEVE: A Pedagogical Agent for Virtual Reality. In: Agents 1998, pp. 332–333. ACM Press, New York (1998)
12. Heylen, D., Nijholt, A., op den Akker, R.: Affect In Tutoring Dialogues. Applied Artificial Intelligence 19(3-4), 287–311 (2005)
13. Hospers, M.A., Kroezen, E., Nijholt, A., op den Akker, R., Heylen, D.: An agent-based intelligent tutoring system for nurse education. In: Applications of Intelligent Agents in Health Care, pp. 143–159. Birkhauser Publishing Ltd., Basel (2003)
14. Poel, M., op den Akker, R., Heylen, D., Nijholt, A.: Emotion based Agent Architectures for Tutoring Systems: The INES Architecture. In: Cybernetics and Systems 2004. Workshop on Affective Computational Entities (ACE 2004), Vienna, pp. 663–667 (2004)
15. Heylen, D., Theune, M., op den Akker, R., Nijholt, A.: Social Agents: the First generations. In: Proceedings of the International Conference on Affective Computing and Intelligent Interaction (2009)
16. Marsella, S., Johnson, W.L., LaBore, C.: Interactive Pedagogical Drama. In: Agents 2000, pp. 301–308. ACM Press, New York (2000)
17. Varni, J.W., Sahler, O.J., Katz, E.R., Mulhern, R.K., Copeland, D.R., Noll, R.B., Phipps, S., Dolgin, M.J., Roghmann, K.: Maternal problem-solving therapy in pediatric cancer. Journal of Psychosocial Oncology 16, 41–71 (1999)
18. Marsella, S., Johnson, W.L.: An Instructor's Assistant for Team-Training in Dynamic Multi-Agent Virtual Worlds. In: Goettl, B.P., Halff, H.M., Redfield, C.L., Shute, V.J. (eds.) ITS 1998. LNCS, vol. 1452, pp. 464–473. Springer, Heidelberg (1998)
19. Baile, W.F., Buckman, R., Lenzi, R., Glober, G., Beale, E.A., Kudelka, A.P.: SPIKES-A six-step protocol for delivering bad news: application to the patient with cancer. The Oncologist 5(4), 302–311 (2000)
20. Orlander, J.D., Graeme Fincke, B., Hermanns, D., Johnson, G.A.: Medical residents' first clearly remembered experiences of giving bad news. Journal Of General Internal Medicine 17(11), 825–840 (2002)
21. Friedrichsen, M.J., Strang, P.M.: Doctors' strategies when breaking bad news to terminally ill patients. J. Palliat. Med. 6(4), 565–574 (2003)

22. Garg, A., Buckman, R., Kason, Y.: Teaching medical students how to break bad news. CMAJ: Canadian Medical Association Journal 156, 1159–1164 (1997)
23. Carver, C.S., Scheier, M.F., Weintraub, J.K.: Assessing coping strategies: A theoretically based approach. Journal of Personality and Social Psychology 56, 267–283 (1989)
24. Folkman, S., Lazarus, R.S.: An analysis of coping in a middle aged community sample. Journal of Health and Social Behavior 21, 219–239 (1980)
25. Kübler-Ross, E.: On Death and Dying, Scribner (1969)
26. Cohen, P.R., Levesque, H.J.: Intention is Choice with Commitment. Artif. Intell. 42(2-3), 213–261 (1990)
27. Rao, A.S., Georgeff, M.P.: Modeling Rational Agents within a BDI-Architecture. In: Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR 1991), pp. 473–484. Morgan Kaufmann Publishers Inc., San Francisco (1991)
28. Wooldridge, M.: Reasoning about Rational Agents. MIT Press, Cambridge (2000)
29. Georgeff, M., Pell, B., Pollack, M., Tambe, M., Wooldridge, M.: The Belief-Desire-Intention Model of Agency. In: Rao, A.S., Singh, M.P., Müller, J.P. (eds.) ATAL 1998. LNCS (LNAI), vol. 1555, pp. 1–10. Springer, Heidelberg (1999)

Intelligent NPCs for Educational Role Play Game

Mei Yii Lim[1], João Dias[2], Ruth Aylett[1], and Ana Paiva[2]

[1] School of Mathematical and Computer Sciences,
Heriot Watt University,
Edinburgh, EH14 4AS, Scotland
{myl, ruth}@macs.hw.ac.uk
[2] INESC-ID, IST, Taguspark,
Av. Prof. Dr. Cavaco Silva,
2744-016 Porto Salvo, Portugal
{joao.dias, ana.paiva}@gaips.inesc-id.pt

Abstract. Video games in general and educational role play games in particular would increase in believability if Non Player Characters reacted appropriately to the player's actions. Realistic and responsive feedback from game characters is important to increase engagement and enjoyment in players. In this paper, we discuss the modelling of autonomous characters based on a biologically-inspired theory of human action regulation taking into account perception, motivation, emotions, memory, learning and planning. These agents populate an educational Role Playing Game, ORIENT (Overcoming Refugee Integration with Empathic Novel Technology) dealing with the cultural-awareness problem for children aged 13 to 14.

1 Introduction

Non Player Characters (NPCs) vary in importance and may play roles of bystanders, allies or competitors to the player in the fictional world of computer games. These NPCs' behaviour is usually scripted and automatic, triggered by certain actions of or dialogue with the player. This method is cumbersome and produces gameplay that is repetitive and thus unnatural. In more advanced Computer Role Playing Games (CRPGs), NPC behaviour can be more complex and players' choices may affect the course of the game, as well as the conversation (eg. Fallout3[1]). However, true dialogues with NPCs are still a problem. In most CRPGs, the same dialogue option chosen by the player will usually receive the same reply from the NPC.

It is beneficial for NPCs to be believable and 'real' so that the player will enjoy interacting with them. Characters that are able to express their feelings and can react emotionally to events are more life-like. NPCs capable of dynamically reacting to player actions in reasonable and realistic ways are therefore very

[1] http://fallout.bethsoft.com/eng/home/home.php

desirable. Natural interaction between NPCs and the player is very important because it transforms the challenge of the game from a technical one to an interpersonal one, and thus may increase both the enjoyment and the engagement of players. However, up-to-date, real autonomous agents that are capable of improvisational actions, appear to be able to 'think', and have desires, motivations and goals of their own, are still rare in games.

2 Educational Role Playing Game

Researchers pointed out that play is a primary socialization and learning mechanism common to all human cultures and many animal species. 'Lions do not learn to hunt through direct instruction but through modeling and play.' [1]. Games are effective because learning takes place within a meaningful context where what must be learned is directly related to the environment in which learning and demonstration take place.

How can cultural studies be made exciting? Perhaps through an educational role play game. For instance, one in which the student is a space command member who must master the patterns of behaviour of an alien culture and pass as their friend within a digitally simulated world. The students will have interesting missions to keep them motivated and engaged. This approach shifts the students' cognitive effort from reading about educational content to hands-on experience of achieving compelling goals. Members of a team can cooperate with each other to solve the team's conflicts with other agents, whether a player from another team or an NPC. Such an opponent must be perceivable as endowed with a personality if the player is to be able to suspend disbelief in the way engagement with the storyworld requires.

In ORIENT[2], our game world is designed in just such a way. It is an interactive computer assisted role-playing game where three players act as visitors to a foreign planet that is inhabited by an alien culture. In order to save the planet from an imminent catastrophe, the players have to cooperate with the alien inhabitants, which can only be achieved by integrating themselves into the culture. Since the game incorporates a social setting, each NPC must be able to establish social relationships with other NPCs and the players to ensure successful collaboration. ORIENT characters must be able to recognise cultural differences and use this information to adapt to other cultures dynamically. The ability to empathise, that is, to detect the internal states of others and to share their experience, is vital to the formation of long-term relationships. Since enhancement of integration in a cultural group relies both on the understanding of the internal states of the persons involved and their affective engagement, both cognitive [2] and affective [3] empathy are relevant. Additionally, previous experience is crucial in maintaining long-term social relationships, which means a requirement for an autobiographic memory [4] is inevitable. Through an ability to retrieve previous experiences from its autobiographic memory, an NPC will be able to know how to react sensibly to a similar future situation. Thus,

[2] http://www.e-circus.org/

ORIENT provides a good case study for modelling NPCs with adaptive and improvisational capabilities, that possess autobiographical memory, individual personality and show empathy.

3 Related Work

Much recent work has been carried out on developing agents with autonomous capabilities. Some of this work focuses on physiological aspects while some focuses instead on cognitive aspects of human action regulation. Examples of existing physiological architectures are those by Cañamero [5], Velásquez [6] and Blumberg [7]. Cañamero's architecture relies on both motivations and emotions to perform behaviour selection for an autonomous creature. Velásquez developed a comprehensive architecture of emotion based on Izard's four systems model [8], focusing on the neural mechanism underlying emotional processing. Blumberg developed an animated dog, Silas that has a simple mechanism of action-selection and learning combining the perspective of ethology and classical animation. A more recent implementation of the model is AlphaWolf [9], capturing a subset of the social behaviour of wild wolves. These architectures are useful for developing agents that have only existential needs but are too low level for characters which require planning and storytelling capabilities as in ORIENT. Another problem of these architectures is that the resulting agents do not show emotional responses to novel situations because all behaviours are hard-coded.

On the cognitive end, the OCC cognitive theory of emotions [10] is one of the most used emotion appraisal model in current emotion synthesis systems. The authors view emotions as valenced reactions that result from three types of subjective appraisals: the appraisal of the desirability of events with respect to the agent's goals, the appraisal of the praiseworthiness of the actions of the agent or another agent with respect to a set of standards for behaviour, and the appraisal of the appealingness of objects with respect to the attitudes of the agent. Numerous implementations of the theory aimed at producing agents with a broad set of capabilities, including goal-directed and reactive behaviour, emotional state and social knowledge exist, beginning with the Affective Reasoner architecture [11], the Em component [12] of the Hap architecture [13], EMA [14], FAtiMA (FearNot! Affective Mind Architecture) [15] and many more. On the other hand, most deliberative agent architectures are based on the BDI (Beliefs, Desires, Intentions) model [16]. The ways BDI agents take their decisions, and the reason why they discard some options to focus on others, however, are questions yet to be answered. These problems are associated with the BDI architecture itself and not with a particular instantiation. Furthermore, BDI agents do not learn from errors and experience.

In order to create purely autonomous agents, we argue that a hybrid architecture combining both physiological and cognitive aspects is required. Some examples of this type of architecture are those by Sloman [17], Jones [18], Oliveira [19] and Dörner [20]. The agent cognitive processes should result from lower-level physiological processing and the outcome of cognitive processes should influence the agent's bodily states, producing complex behaviours that can be termed emotional. Damasio [21] provides neurological support for the idea that there is no 'pure reason' in the healthy human brain but emotions are vital for healthy rational human thinking and behaviour which means both cognitive and physiological systems are essential parts of intelligent agents.

4 ORIENT Agent Mind

4.1 Inspiration

The ORIENT agent mind (i.e. the program that controls NPCs' behaviour) is built upon FAtiMA [15] architecture applied in FearNot!v2.0. FAtiMA was an extension of a BDI architecture, hence, faced the problem of ambiguity in its decision making processes common in any BDI architecture. It has a reactive and a deliberative appraisal layer. The reactive appraisal process matches events with a set of predefined emotional reaction rules while the deliberative appraisal layer generates emotions by looking at the state of current intentions, more concretely whether an intention was achieved or failed, or the likelihood of success or failure. After the appraisal phase, both reactive and deliberative components perform practical reasoning. The reactive layer uses simple and fast action rules that trigger action tendencies. On the other hand, the deliberative layer uses the strength of emotional appraisal that relies on importance of success and failure of goals for intention selection. A goal is activated only if its start conditions are satisfied. Each goal also contains success and failure conditions.

The main reason for choosing FAtiMA is that it incorporates the OCC theory [10], has a continous planner [22] that is capable of partial order planning and includes both problem-focused and emotion-focused coping [23] in plan execution. The OCC theory models empathy easily because it takes into consideration appraisals of events regarding the consequences for others. It is - as far as we know - the only model that provides a formal description of non-parallel affective empathic outcomes (i.e. emotions that take a bad relationship between one agent and another into account, e.g., gloating and resentment). Moreover, since the OCC model includes emotions that concern behavioural standards and social relationships based on like/dislike, praiseworthiness and desirability for others, it allows appraisal processes that take into consideration cultural and social aspects, important for ORIENT agents.

However, the number of empathic emotional outcomes described in OCC: happy-for, resentment, gloating and pity is limited. Moreover, FAtiMA does not take the physiological aspects of emotion into account. Another problem with FAtiMA is the tedious authoring process of the character's goals, emotional reactions, actions and effects, and action tendencies so that the final behaviour of the characters is as intended. Having these values scripted reduces the dynamism of some of the core aspects modeled, resulting in agents that are not adaptive and do not learn from experience.

To address these constraints, we considered the PSI theory [20], a psychologically-founded theory that incorporates all the basic components of human action regulation: perception, motivation, cognition, memory, learning and emotions. It allows for modelling autonomous agents that adapt their internal representations to a dynamic environment. A few successes of the PSI model in replicating human behaviour in complex tasks can be found in [20, 24]. A PSI agent does not require any executive structure that conducts behaviour, rather, processes are self-regulatory and run in parallel driven by needs. Memory functions as a central basis for coordination.

Emotions within the PSI theory are conceptualised as specific modulations of cognitive and motivational processes enabling a wide range of empathic emotional effects. These modulations are realised by *emotional parameters. Arousal* is the preparedness for perception and reaction; *resolution level* determines the accuracy of cognitive processes; and *selection threshold* prevents oscillating between behaviours by giving the current intention priority. Different combinations of these parameter values lead to different physiological changes that resemble emotional experiences in biological agents. For example, if an event leads to a drop in the character's certainty, then its *arousal* level increases causing a decrease in the *resolution level.* In such situation, a quick reaction is required hence forbidding time-consuming search. The character will concentrate on the task in order to recover the deviated need(s) and hence may choose to carry out the first action that it found feasible. The character may be diagnosed as experiencing anxiety. Therefore, depending on the cognitive resources and the motivational state of the agent in a given situation, these parameters are adjusted, resulting in more or less careful or forceful ways of acting, as well as more or less deliberate cognitive processing.

Since, FAtiMA already includes perception, cognition, memory and emotions, we added the PSI motivational and learning components into the existing architecture. The motivational system serves as a quick adaptation mechanism of the agent to a specific situation and may lead to a change of belief about another agent as shown in [25], important for conflict resolution among ORIENT characters. PSI's other advantage over FAtiMA is that it does not require much authoring except initialising the agents with some prior knowledge. PSI agents' differences in behaviour will then correspond to different life-experiences that lead to different learned associations. Thus, PSI permits more flexibility both in authoring and the characters' behaviour than FAtiMA. Unfortunately, this also means lack of control over the characters' behaviour which is a problem because characters in ORIENT need to behave in certain ways so that the educational goal can be reached. According to [26], good educational games are games where narrative events situate the activity, constraining actions, provoking thought and sparking emotional responses. By making the NPCs react in certain ways, the player's ability to access information or manipulate the world is limited. This forces the player to evaluate the relative value of information and to devise appropriate goals and strategies to resolve complex authentic problems and help them to develop an experiential understanding of what might be otherwise an abstract principle.

Combining FAtiMA and PSI, the problems of psychological plausibility and control are addressed, neither of which can be solved by either architecture alone. Cultural and social aspects of interaction can be modelled using FAtiMA while PSI provides an adaptive mechanism for action regulation, fulfilling the requirements of ORIENT characters. Author are free to decide how much information they want to provide the characters to start with and leave the rest for the characters to learn. The degree of desirability (or undesirability) of an action or event is proportionate to the degree of positive (or negative) changes that an action or event brings to the agent's drives. This desirability value can be used to automatically generate emotions according to the OCC model, removing some of the need to write predefined domain-specific emotional reaction rules. This means that the reactive layer in FAtiMA may be omitted.

4.2 FAtiMA-PSI Architecture

In the ORIENT agent mind architecture shown in Figure 1, goals are driven by needs. A motivational system as in PSI provides the character with a basis for selective attention, critical for learning and memory processes, hence increases its adaptive prowess. Five basic drives from PSI are modeled in ORIENT including Energy, Integrity, Affiliation, Certainty and Competence. These needs can emerge over time or can be activated by events happening in the environment. Energy represents an overall need to preserve the existence of the agent (food + water). As the agent carries out actions, it consumes energy which means that eventually, it will have to rest or perform actions to regain energy. Integrity represents well being, i.e. the agent avoids pain or physical damage while affiliation is useful for social relationships. On the other hand, certainty and competence influence cognitive processes. It is assumed that the scales for all drives are comparable, ranging from 0 to 10 where 0 means complete deprivation while 10

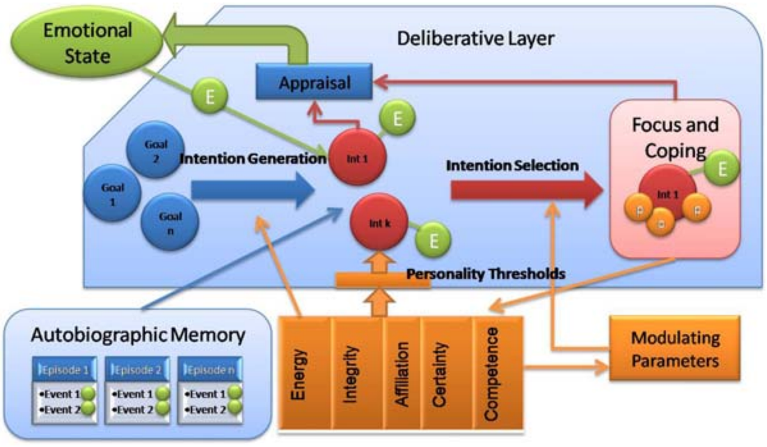


Fig. 1. FAtiMA-PSI architecture

means complete satisfaction. An agent's aim is to maintain these drives at the highest level possible at all time in order to function properly.

The motivational system also allows the creation of agents with personality. Each drive has a specific weight ranging from 0 to 1 that underlines its importance to an agent. The strength of a drive (*Strength(d)*) depends on its current strength plus the amount of deviation from the set point (effect of goal/action) and the specific weight of the drive. For example, if agent A is a friendly character, affiliation would be an important factor in its social relations, say weight 0.8 while a hostile agent B would have a low importance for affiliation, say weight 0.3. Now, if both agents have a current affiliation value of 2 and if the deviation from set point is -4, agent A's strength for affiliation would be -1.2 (2+(-4*0.8)) while agent B's strength for affiliation would be 0.8 (2+(-4*0.3)) based on Equation 1. The higher the strength of a drive, the lower the agent's need is for that particular drive. In this case, agent A will work harder to satisfy its need for affiliation than agent B. So, by assigning different weights for different needs to different agents, characters with different personalities can be produced.

$$Strength(d) = Strength(d) + (Deviation(d) * Weight(d)) \tag{1}$$

A goal is define by the following attributes:

- Id: the goal identifier or name
- Preconditions: a list of conditions that determine when the goal becomes active
- SuccessConditions: a list of conditions used to determine if the goal is successful
- FailureConditions: a list of conditions that determine the goal failure
- EffectsOnDrives: specifies the effects that the goal will have on the agent's drives if the goal succeeds

Each goal contains information about its expected contribution to energy, integrity and affiliation, that is, how much the drives may be deviated from or satisfied if the goal is performed. Likewise, events or actions also include contributions to drives. Based on this information, the importance of goals to each character at a particular time instance can be determined, allowing the character to give priority to goals that satisfy its needs under different circumstances. This is an advantage over the previous FAtiMA architecture where a goal's importance is pre-authored which mean that whenever a goal activation condition becomes true, the goal is always created with the same importance of success and failure, independently of the situation that originated the goal. This causes a problem in deciding which goal should be selected when there are several conflicting goals. The effects of needs are also useful in the appraisal phase to create emotional impact that will be stored in the autobiographic memory and guide the agent's further actions. Since each agent has a different personality, the effect of an event may differ from one agent to another, which in turn affects their emotional and behavioural responses. Thus, needs can be considered both the source of behaviour and feedback from the effect of behaviour, a fundamental aspect necessary for learning agents.

As for certainty and competence, no explicit specification of contributions to these is necessary because they are cognitive needs and their values can be calculated automatically as described below. Whenever an expected event fails to turn up or an unknown object appears, the agent's certainty drops. Thus, uncertainty represents the extent to which knowledge about a given fact/action is not accurate or not known. We model uncertainty using error prediction with an Exponential Moving Average where the weighting factors decreases exponentially resulting in the lastest data being the most important. ORIENT characters continuously make predictions about the probability of success of their goals. These predictions are then compared with the actual outcomes. The difference between these two values is the *ObservedError*. Based on past observed errors, we can estimate the current error, that is, the current uncertainty using Equation 2. α represents the rate past observations lose importance and t is the time step for the character's mind cycle.

$$Uncertainty(t) = \alpha * ObservedError(t-1) + (1-\alpha) * Uncertainty(t-1) \quad (2)$$

Hence, gaining certainty is not about avoiding uncertain goals but trying out these goals. This is because in order to achieve certainty, the character has to reduce the estimation error, and the goals which have high error estimations are goals that contribute more to certainty. Certainty is achieved by exploration of new strategies or actions, which leads to the construction of more complete hypotheses. If trial and error is too dangerous, developments in the environment are observed in order to collect more information. By doing so, the character can change its behaviour dynamically. Please note that the character does not learn by forming new goals because this will lead to a lack of control over its behaviour. Instead, it learns by trying out different actions from a pre-specified set of actions and remembering which actions helped it to tackle a situation best. This information is stored in its autobiographic memory and serves as an indicator to the success probability of satisfying a specific need in future. Since certainty depends on the amount of unkown information relating to a goal, the more an agent encounters the same type of situation, the higher its certainty is regarding the situation.

Competence represents the efficiency of an agent in reaching its goals and fulfilling its demands. Success increases competence while failure decreases it. The agent's autobiographic memory provides a history of previous interactions, which records the agent's experience in a task (the number of successes in performing a goal) useful for the calculation of goal competence (likelihood of success in performing a goal, Equation 3). Since no distinction is made in calculating competence between achieving an important goal or a less important one, one can assume that all goals have the same contribution to the success rate. If the agent cannot remember previous activations of the goal, then it ignores the likelihood of success and increases the goal's contribution to certainty.

$$Comp(g) = NoOfSuccesses(g)/NoOfTries(g) \quad (3)$$

$$OverallComp = NoOfSuccesses/NoOfGoalsPerformed \quad (4)$$

The autobiographic memory also stores information about the agent's overall performance (the number of successes so far taking into consideration all goals performed) useful for the calculation of overall competence (Equation 4). The expected competence (Equation 5) of the agent will then be a sum of its overall competence and its competence in performing a current goal. A low competence level indicates that the agent should avoid taking risks and choose options that have worked well in the past. A high competence means that the agent can actively seek difficulties by experimenting with new courses of action that are less likely to succeed. Together, competence and certainty direct the agent towards explorative behavior; depending on its abilities and the difficulty of mastering the environment, it will actively seek novelty or avoid complexity. During this learning process, the agent also remembers any specific expressed emotions by other agents in particular situations. It continuously updates and adapts this information enabling empathic engagement in future interactions.

$$ExpComp(g) = OverallComp + Comp(g) \tag{5}$$

At the start of an interaction, each agent has a set of initial values for needs. Based on the level of its current needs, the agent generates intentions, that is, it activates goal(s) that are relevant to the perceived circumstances. A need may have several goals that satisfy it (e.g. I can gain affiliation by making a new friend or socialising with an old friend) and a goal can also affect more than one need (e.g. eating food offered by another agent satisfies the need for energy as well as affiliation). So, when determining a goal's strength (Equation 6), all drives that it satisfies are taken into account. A goal that satisfies more drives will have a higher strength than those that satisfy less.

$$Strength(g) = \sum Strength(d) \tag{6}$$

For a particular need, the more a goal reduces its deviation, the more important that goal is (e.g. eating a full carbohydrate meal when you're starving satisfies you better than eating a vegetarian salad). By looking at the contribution of the goal to overall needs and to a particular need, goals that satisfy the same need can be compared so that success rate in tackling the current circumstances can be maximised. So, the utility value of a goal can be determined taking into consideration overall goal strength on needs (*Strength(g)*), contribution of the goal to a particular need (*ExpCont(g, d)*) and the expected competence (*ExpComp(g)*) of the agent. Additionally, the urgency of a goal is taken into account. *Urgency(g)* gives importance to goals that should become active immediately, usually goals that satisfy the most current deviated need(s).

$$EU(g) = (1+goalUrgency(g)) * ExpComp(g) * Strength(g) * ExpCont(g, d) \tag{7}$$

On each cycle, goals are checked to see if any has become active by testing the goal's preconditions. Once a goal becomes active, a new intention to achieve the goal is created and added to the intention structure. The intention represents the agent's commitment to achieve the goal and stores all plans created for it.

Since there can be more than one intention activated at any particular time instance, the character must choose one of them to continue deliberation (and planning). Applying PSI, the selection of goal in ORIENT is performed based on the *selection threshold* value. The current active intention is selected based on a winner takes all approach, that is, the goal with the highest expected utility value is chosen. An unselected goal can be activated if its strength exceeds the value of the current active intention multiply by the *selection threshold*. So, if the *selection threshold* is high, it is less likely for another goal to be activated, hence, allowing the agent to concentrate on its current active intention. After an intention is selected, the agent proceeds to generate plan(s) to achieve it.

When a plan is brought into consideration by the reasoning process, it generates and updates OCC prospect based emotions such as:

– Hope: Hope to achieve the intention. The emotion intensity is determined from the goal's importance of success and the plan's probability of success.
– Fear: Fear of not being able to achieve the intention. The emotion intensity is determined from the goal's importance of failure and the plan's probability of failing.

All active goals are then checked to determine if the goal succeeds or fails. If the planner is unable to make a plan, more prospect based emotions will be generated, such as Satisfaction, Disappointment, Relief and Fears-Confirmed. In order to cope with different circumstances, ORIENT characters perform two types of coping: problem-focused coping and emotion-focused coping as in FAtiMA. Problem-focused coping focuses on acting on the environment to tackle a situation. It involves planning a set of actions that achieve a desired result and executing those actions. On the other hand, emotion-focused coping works by changing the agent's interpretation of circumstances, that is, lowering strong negative emotions for example, by lowering the importance of goals, a coping strategy used often by people when problem focused coping has low chances of success. These coping strategies are triggered by emotions and personality of the characters. For instance, a fearful character has a higher chance to drop an uncertain goal than a hopeful character.

5 Conclusion and Future Work

In this paper, we discussed our effort in developing autonomous NPCs for an educational role play game. In ORIENT, characters behaviour is regulated by a biologically-inpired architecture of human action regulation. The new addition of the motivational system onto FAtiMA provides ORIENT characters with a basis for selective attention, critical for learning and memory processes. Intentions are selected based on strength of activated needs, urgency and success probability addressing the BDI architecture ambiguity in decision making. The resulting agents learn through trial and error, allowing more efficient adaptation and empathic engagement in different social circumstances. The successful linking of body and mind is consistent with that of humans' and hence, should produce characters with behaviours that seem plausible to a human. The software which is written in Java has been made available at the open source portal SourceForge[3] and is reusable in autonomous agents applications.

Acknowledgements

This work was partially supported by European Commission (EC) and is currently funded by the eCIRCUS project IST-4-027656-STP with university partners Heriot-Watt, Hertfordshire, Sunderland, Warwick, Bamberg, Augsburg, Wuerzburg plus INESC-ID and Interagens. The authors are solely responsible for the content of this publication. It does not represent the opinion of the EC, and the EC is not responsible for any use that might be made of data appearing therein.

References

[1] Eck, R.V.: Digital game-based learning: It's not just the digital natives who are restless. EDUCAUSE Review 41(2), 16–30 (2006)
[2] Hogan, R.: Development of an empathy scale. Journal of Consulting and Clinical Psychology (35), 307–316 (1977)
[3] Hoffman, M.L.: Empathy, its development and prosocial implications. Nebraska Symposium on Motivation 25, 169–217 (1977)
[4] Ho, W.C., Dautenhahn, K., Nehaniv, C.L.: Computational memory architectures for autobiographic agents interacting in a complex virtual environment: A working model. Connection Science 20(1), 21–65 (2008)
[5] Cañamero, D.: A hormonal model of emotions for behavior control. In: VUB AI-Lab Memo 97-06, Vrije Universiteit Brussel, Belgium (1997)
[6] Velásquez, J.D.: Modeling emotions and other motivations in synthetic agents. In: Proceeding AAAI 1997, pp. 10–15. AAAI Press and The MIT Press (1997)
[7] Blumberg, B.: Old Tricks, New Dogs: Ethology and Interactive Creatures. PhD thesis, Massachusetts Institute of Technology. MIT, Cambridge (1996)
[8] Izard, C.E.: Four systems for emotion activation: Cognitive and noncognitive processes. Psychological Review 100(1), 68–90 (1993)
[9] Tomlinson, B., Blumberg, B.: Alphawolf: Social learning, emotion and development in autonomous virtual agents. In: First GSFC/JPL Workshop on Radical Agent Concepts, October 4 (2002)
[10] Ortony, A., Clore, G., Collins, A.: The cognitive structure of emotions. Cambridge University Press, Cambridge (1988)
[11] Elliot, C.D.: The Affective Reasoner: A process model of emotions in an multi-agent system. PhD thesis, Northwestern University, Illinios (1992)
[12] Reilly, W.S., Bates, J.: Building emotional agents. Technical Report CMU-CS-91-143, School of Computer Science, Carnegie Mellon University (1992)
[13] Loyall, A.B., Bates, J.: Hap: A reactive adaptive architecture for agents. Technical Report CMU-CS-91-147, School of Computer Science, Carnegie Mellon University (1991)

[3] http://sourceforge.net/projects/orient-ecircus

[14] Gratch, J., Marsella, S.: Evaluating a computational model of emotion. Journal of Autonomous Agents and Multiagent Systems (Special issue on the best of AAMAS 2004) 11, 23–43 (2004)
[15] Dias, J., Paiva, A.: Feeling and reasoning: A computational model for emotional agents. In: Bento, C., Cardoso, A., Dias, G. (eds.) EPIA 2005. LNCS (LNAI), vol. 3808, pp. 127–140. Springer, Heidelberg (2005)
[16] Bratman, M.E.: Intention, Plans and Practical Reasoning. Harvard University Press, Cambridge (1987)
[17] Sloman, A.: Varieties of affect and the cogaff architecture schema. In: Symposium on Emotion, Cognition and Affective Computing, AISB 2001 Convention, March 21-24. University of York, United Kingdom (2001)
[18] Jones, R.M., Henninger, A.E., Chown, E.: Interfacing emotional behavior moderators with intelligent synthetic forces. In: Proceeding of the 11th CGF-BR Conference, Orlando, FL, May 7 (2002)
[19] Oliveira, E., Sarmento, L.: Emotional advantage for adaptability and autonomy. In: AAMAS, Melbourne, Australia, pp. 305–312. ACM, New York (2003)
[20] Dörner, D.: The mathematics of emotions. In: Frank Detje, D.D., Schaub, H. (eds.) Proceedings of the Fifth International Conference on Cognitive Modeling, Bamberg, Germany, April 10-12, pp. 75–79 (2003)
[21] Damasio, A.: Descartes' Error: Emotion, Reason and the Human Brain. Gosset/Putnam Press, New York (1994)
[22] Aylett, R., Dias, J., Paiva, A.: An affectively driven planner for synthetic characters. In: International Conference on Automated Planning and Scheduling (ICAPS 2006), UK (2006)
[23] Marsella, S., Johnson, B., LaBore, C.: Interactive pedagogical drama. In: Fourth International Conference on Autonomous Agents (AAMAS), Bologna, Italy, pp. 301–308. ACM Press, New York (2002)
[24] Dörner, D., Gerdes, J., Mayer, M., Misra, S.: A simulation of cognitive and emotional effects of overcrowding. In: Proceedings of the 7th International Conference on Cognitive Modeling, Trieste, Italy, April 5-8 (2006)
[25] Lim, M.Y.: Emotions, Behaviour and Belief Regulation in An Intelligent Guide with Attitude. PhD thesis, School of Mathematical and Computer Sciences, Heriot-Watt University, Ediburgh, Edinburgh (2007)
[26] Squire, K.: Design principles of next generation digital gaming for education. Educational Technology 43(5), 17–33 (2003); The Games-To-Teach Team at MIT

Design of a Decision Maker Agent for a Distributed Role Playing Game – Experience of the SimParc Project

Jean-Pierre Briot[1], Alessandro Sordoni[1], Eurico Vasconcelos[2], Marta de Azevedo Irving[3], Gustavo Melo[3], Vinícius Sebba-Patto[1], and Isabelle Alvarez[1]

[1] Laboratoire d'Informatique de Paris 6 (LIP6), Université Pierre et Marie Curie – CNRS, Paris, France
[2] Computer Science Department, Pontifícia Universidade Católica (PUC-Rio), Rio de Janeiro, RJ, Brazil
[3] EICOS Program, Universidade Federal do Rio de Janeiro (UFRJ), Rio de Janeiro, RJ, Brazil

Abstract. This paper addresses an ongoing experience in the design of an artificial agent taking decisions in a role playing game populated by human agents and by artificial agents. At first, we will present the context, an ongoing research project aimed at computer-based support for participatory management of protected areas (and more specifically national parks) in order to promote biodiversity conservation and social inclusion. Our applicative objective is, through a distributed role-playing game, to help various stakeholders (e.g., environmentalist, tourism operator) to collectively understand conflict dynamics for natural resources management and to explore negotiation management strategies for the management of parks. Our approach includes support for negotiation among players and insertion of various types of artificial agents (decision making agent, virtual players, assistant agents). In this paper, we will focus on the architecture of the decision making agent playing the role of the park manager, the rationales for its decision, and how it takes into account the preferences/votes from the stakeholders.

1 Introduction

In this paper, we are discussing our experience of inserting artificial agents in a role-playing game populated with humans. The role playing game we consider may be considered as a serious game, as our objective is educational and epistemic. In this game, humans play some role and discuss, negotiate and take decisions about a common domain, in our case environment management decisions.

We are currently designing and inserting different types of artificial agents into this human-based role-playing game. More precisely, we are considering three types of artificial agents:

– artificial players – A first motivation is to address the possible absence of sufficient number of human players for a game session [1]. But this will also allow more systematic experiments about specific configurations of players profiles, because of artificial players' objective, deterministic and reproducible behaviors.
– artificial decision maker – The park manager acts as an arbitrator in the game, making a final decision for types of conservation for each landscape unit and it also explains its decision to all players. As for artificial players, using an artificial manager in place of a human manager will allow reproductible experiments with controllable levels of participation and of manager profile (see Section 6.1).
– assistant agents – These agents are designed to assist a player by performing tasks, such as orientation within games steps and actions expected, and also support for negotiation, e.g., by identifying and suggesting potential coalitions.

In this paper we focus on the design of the artificial decision maker agent. Its objective is to take decision based on its own analysis of the situation and on the proposals by the players. The agent is also able to explain its decision based on its chain of argumentation.

The structure of this paper is as following: after introducing the SimParc project, its role playing game and its computer support, and the insertion of artificial agents, we describe the decision maker agent objectives, architecture and implementation.

2 The SimParc Project

2.1 Project Motivation

A significant challenge involved in biodiversity management is the management of protected areas (e.g., national parks), which usually undergo various pressures on resources, use and access, which results in many conflicts. This makes the issue of conflict resolution a key issue for the participatory management of protected areas. Methodologies intending to facilitate this process are being addressed via bottom-up approaches that emphasize the role of local actors. Examples of social actors involved in these conflicts are: park managers, local communities at the border area, tourism operators, public agencies and NGOs. Examples of inherent conflicts connected with biodiversity protection in the area are: irregular occupation, inadequate tourism exploration, water pollution, environmental degradation and illegal use of natural resources.

Our SimParc project focuses on participatory parks management. (The origin of the name SimParc stands in French for "Simulation Participative de Parcs") [2]. It is based on the observation of several case studies in Brazil. However, we chose not to reproduce exactly a real case, but a fictive park, in order to leave the door open for broader game possibilities [3]. Our project aim is to help various stakeholders at collectively understand conflicts and negotiate strategies for handling them.

2.2 Approach

Our initial inspiration is the companion modeling (ComMod) approach about participatory methods to support negotiation and decision-making for participatory management of renewable resources [4]. They pioneer method, called MAS/RPG, consists in coupling multi-agent simulation (MAS) of the environment resources and role-playing games (RPG) by the stakeholders [4]. The RPG acts like a "social laboratory", because players of the game can try many possibilities, without real consequences.

Recent works proposed further integration of role-playing into simulation, and the insertion of artificial agents, as players or as assistants. Participatory simulation and its incarnation, the Simulación framework [5], focused on a distributed support for role-playing and negotiation among human players. All interactions are recorded for further analysis (thus opening the way to automated acquisition of behavioral models) and assistant agents are provided to assist and suggest strategies to the players. The Games and Multi-Agent-based Simulation (GMABS) methodology focused on the integration of the game cycle with the simulation cycle [1]. It also innovated in the possible replacement of human players by artificial players. One of our objectives is to try to combine their respective merits and to further explore possibilities of computer support.

3 The SimParc Role-Playing Game

3.1 Game Objectives

Current SimParc game has an epistemic objective: to help each participant discover and understand the various factors, conflicts and the importance of dialogue for a more effective management of parks. Note that this game is not (or at least not yet) aimed at decision support (i.e., we do not expect the resulting decisions to be directly applied to a specific park).

The game is based on a negotiation process that takes place within the park council. This council, of a consultative nature, includes representatives of various stakeholders (e.g., community, tourism operator, environmentalist, non governmental association, water public agency...). The actual game focuses on a discussion within the council about the "zoning" of the park, i.e. the decision about a desired level of conservation (and therefore, use) for every sub-area (also named "landscape unit") of the park (remember that this is a fictive park, see Section 2.1). We consider nine pre-defined potential levels (that we will consider as types) of conservation/use, from more restricted to more flexible use of natural resources, as defined by the (Brazilian) law. Examples are: Intangible, the most conservative use, Primitive and Recuperation.

The map of the park (a GIS with different layers of information) is shown at Figure 1. It includes icons for various types of "elements of concern" for the different stakeholders (e.g., various animals to be protected, tourism spots, types of illegal occupation, etc.).

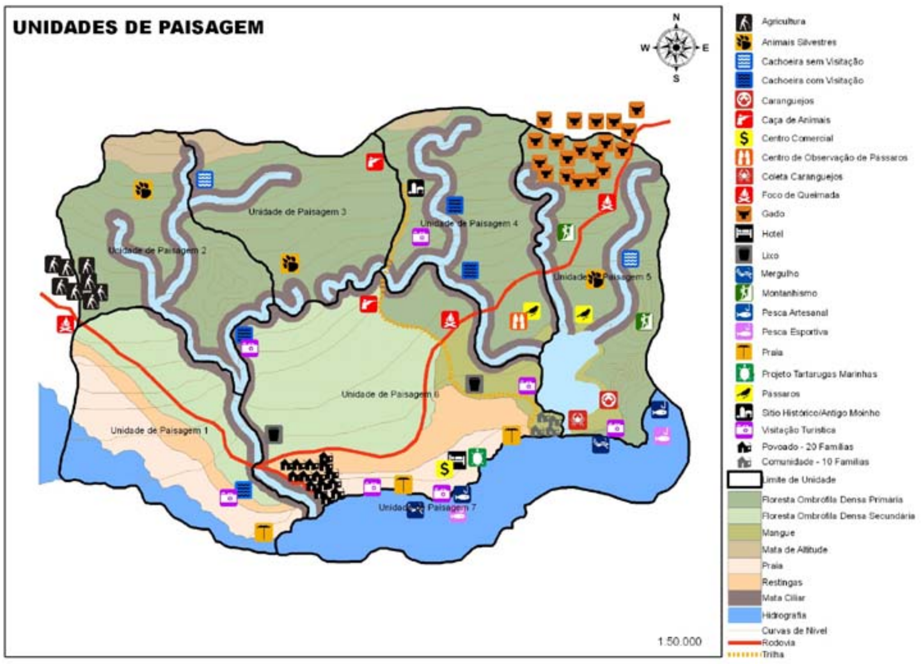


Fig. 1. The map of the SimParc park

The game considers a certain number of players' roles, each one representing a certain stakeholder. Depending on its profile and the elements of concerns in each of the landscape units (e.g., tourism spot, people, endangered species...), each player will try to influence the decision about the type of conservation for each landscape unit. It is clear that conflicts of interest will quickly emerge, leading to various strategies of negotiation (e.g., coalition formation, trading mutual support for respective objectives, etc).

A special role in the game is the park manager. He is a participant of the game, but as an arbiter and decision maker, and not as a direct player. He observes the negotiation taking place among players and takes the final decision about the types of conservation for each landscape unit. His decision is based on the legal framework, on the negotiation process among the players and on his personal profile (e.g., more conservationist or more open to social concerns) [3]. He may also have to explain his decision, if the players so demand. We plan that the players and the park manager may be played by humans or by artificial agents.

3.2 Game Cycle

The game is structured along six steps, as illustrated at Figure 2. At the beginning (step 1), each participant is associated with a role. Then, an initial scenario is presented to each player, including the setting of the landscape units, the

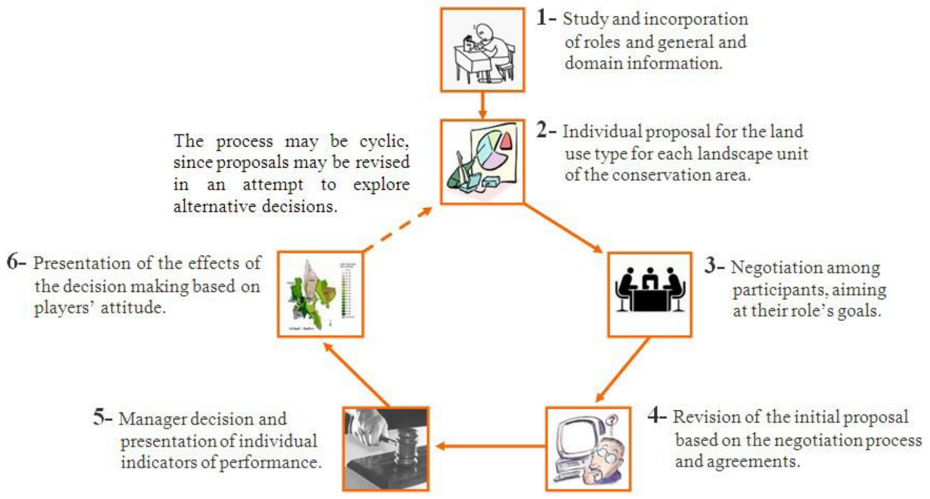

1- Study and incorporation of roles and general and domain information.
2- Individual proposal for the land use type for each landscape unit of the conservation area.
The process may be cyclic, since proposals may be revised in an attempt to explore alternative decisions.
3- Negotiation among participants, aiming at their role's goals.
6- Presentation of the effects of the decision making based on players' attitude.
4- Revision of the initial proposal based on the negotiation process and agreements.
5- Manager decision and presentation of individual indicators of performance.

Fig. 2. The six steps of the SimParc game cycle

possible types of use and the general objective associated with his role. Then (step 2), each player decides a first proposal of types of use for each landscape unit, based on his/her understanding of the objective of his/her role and on the initial setting. Once all players have done so, each player's proposal is made public.

In step 3, players start to interact and to negotiate on their proposals. This step is, in our opinion, the most important one, where players collectively build their knowledge by means of an argumentation process. In step 4, they revise their proposals and commit themselves to a final proposal for each landscape unit. In step 5, the park manager makes the final decision, considering the negotiation process, the final proposals and also his personal profile (e.g., more conservationist or more sensitive to social issues). Each player can then consult various indicators of his/her performance (e.g., closeness to his initial objective, degree of consensus, etc.). He can also ask for an explanation about the park manager decision rationales.

The last step (step 6) "closes" the epistemic cycle by considering the possible effects of the decision. In the current game, the players provide a simple feedback on the decision by indicating their level of acceptance of the decision.[1]

A new negotiation cycle may then start, thus creating a kind of learning cycle. The main objectives are indeed for participants: to understand the various factors

[1] A future plan is to introduce some evaluation of the quality of the decision. Possible alternatives are: computable indicators (e.g., about the economical or social feasibility), a more formal model (based on viability theory, which allows to compute the viability and resilience of the result of the zoning decision), or a multi-agent simulation of the evolution of resources. Note that a real/validated model is not necessary as the park is fictive and the objective is credibility, not realism.

and perspectives involved and how they are interrelated; to negotiate; to try to reach a group consensus; and to understand cause-effect relations based on the decisions.

4 The SimParc Game Support Architecture

4.1 Design and Implementation of the Architecture

Our current prototype benefited from our previous experiences (game sessions and a first prototype) and has been based on a detailed design process. Based on the system requirements, we adopted Web-based technologies (more precisely J2E and JSF) that support the distributed and interactive character of the game as well as an easy deployment.

Figure 3 shows the general architecture and communication structure of SimParc prototype version 2. In this second prototype, distributed users (the players and the park manager) interact with the system mediated internally by communication broker agents (CBA). The function of a CBA is to abstract the fact that each role may be played by a human or by an artificial agent. A CBA also translates user messages in http format into multi-agent KQML format and vice versa. For each human player, there is also an assistant agent offering assistance during the game session. During the negotiation phase, players (human or artificial) negotiate among themselves to try to reach an agreement about the type of use for each landscape unit (sub-area) of the park.

A Geographical Information System (GIS) offers to users different layers of information (such as flora, fauna and land characteristics) about the park geographical area. All the information exchanged during negotiation phase, namely users'

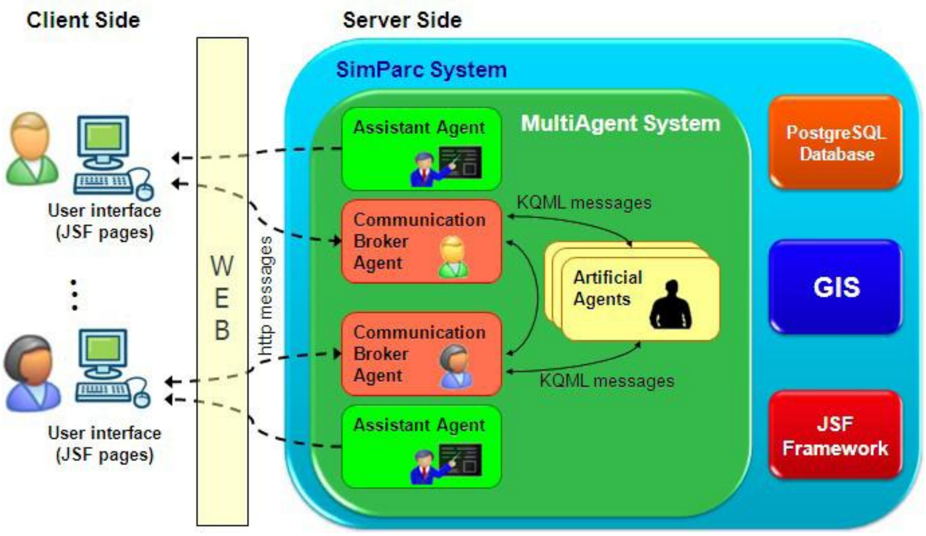


Fig. 3. SimParc version 2 general architecture

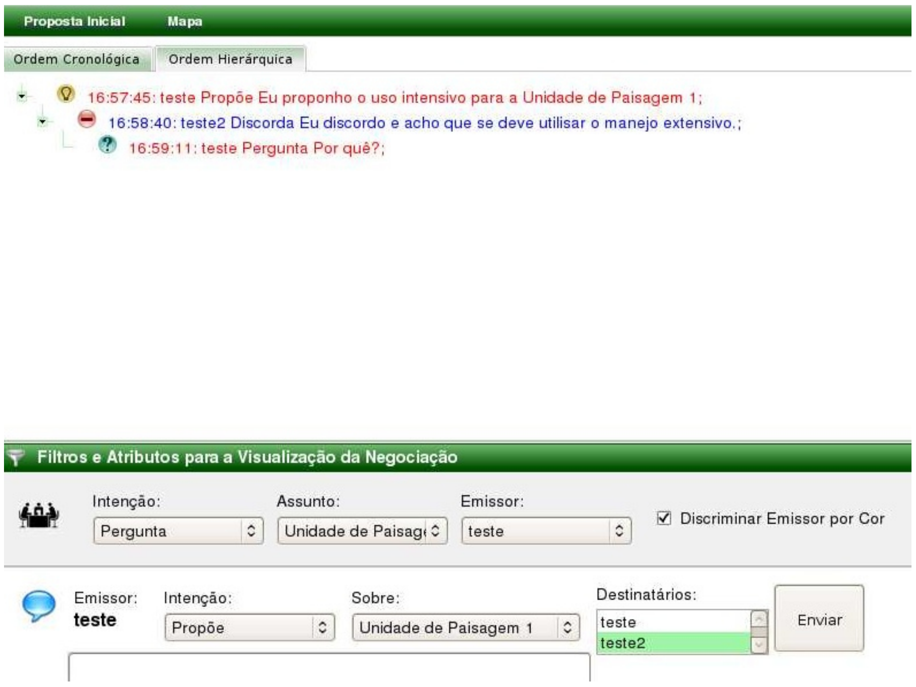
logs, game configurations, game results and general management information are recorded and read from a PostgreSQL database.

4.2 Interface Support

The interface for negotiation is shown at Figure 4. It includes advanced support for negotiation (rhetorical markers and dialogue filtering/structuring mechanisms, see details in [6]), access to different kinds of information about other players, land, law and the help of a personal assistant. The interface for players decision about the types of use is shown at Figure 5. In this interface, the players can analyze the area based in its different layers (e.g., land, hydrography, vegetation. . .).

4.3 Preliminary Evaluation

The current computer prototype has been tested through two game sessions by domain expert players in January 2009. The 9 roles of the game and the park manager were played by humans, Among them, 8 were experts in park management (researchers and professionals, one being a professional park manager in Brazil). The two remaining players were not knowledgeable in park management, one being experienced in games (serious games and video games) and the other one a complete beginner in all aspects.

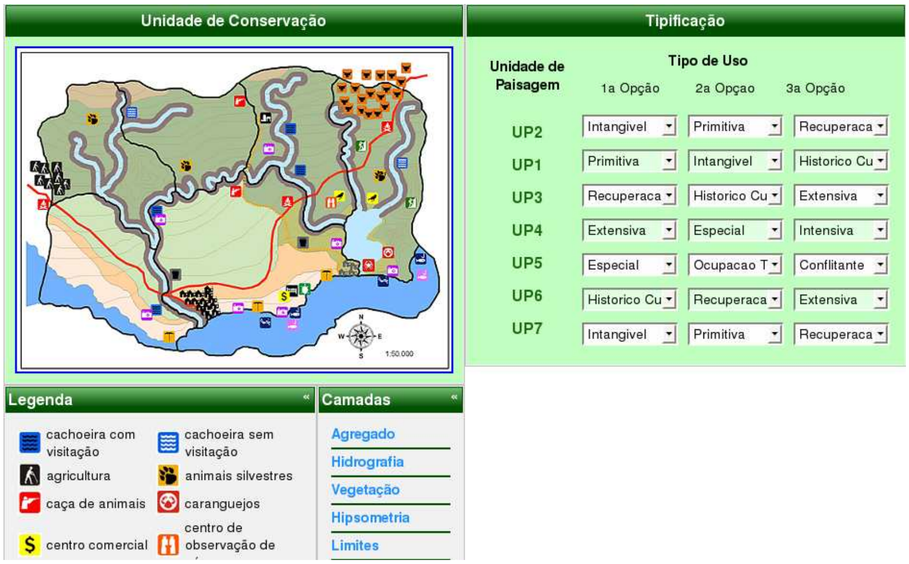


16:57:45: teste Propõe Eu proponho o uso intensivo para a Unidade de Paisagem 1;
16:58:40: teste2 Discorda Eu discordo e acho que se deve utilisar o manejo extensivo.;
16:59:11: teste Pergunta Por quê?;

Intenção: Pergunta
Assunto: Unidade de Paisag
Emissor: teste
☑ Discriminar Emissor por Cor

Emissor: teste
Intenção: Propõe
Sobre: Unidade de Paisagem 1
Destinatários: teste, teste2

Fig. 4. Current prototype's negotiation graphical user interface


Unidade de Conservação

Tipificação

Unidade de Paisagem	Tipo de Uso 1a Opção	2a Opçao	3a Opção
UP2	Intangivel	Primitiva	Recuperaca
UP1	Primitiva	Intangivel	Historico Cu
UP3	Recuperaca	Historico Cu	Extensiva
UP4	Extensiva	Especial	Intensiva
UP5	Especial	Ocupacao T	Conflitante
UP6	Historico Cu	Recuperaca	Extensiva
UP7	Intangivel	Primitiva	Recuperaca

Legenda

cachoeira com visitação	cachoeira sem visitação
agricultura	animais silvestres
caça de animais	caranguejos
$ centro comercial	centro de observação de

Camadas

- Agregado
- Hidrografia
- Vegetação
- Hipsometria
- Limites

Fig. 5. Current prototype's decision graphical user interface

We analyzed data on the game sessions (written questionnaires, recorded debriefing, etc.) and a more detailed analysis is presented in [7]. Overall, the game was well evaluated by the human players. As one of the player also took part in a previous game session in September 2007, with no computer support yet, we could also have some preliminary clues at the benefits of a computer support as well as the relative loss in modality of interactions between players. That player acknowledged the progress in structuring and analysis of the negotiation, thanks to the computer support. An interesting finding after the sessions was also that all players learned and took benefit of the game. The experts explored and refined strategies for negotiation and management, whereas the beginner player took benefit of the game as a more general educational experience about environmental management. In other words, the game appeared to be tolerant to the actual level of expertise of players, an aspect which had not been planned ahead. Last, the fact that the prototype is accessible through Internet via a standard Web browser, and that it does not require preliminary installation of a software on all players computers, was acknowledged as a potential for greater mobility and larger dissemination of this game. In summary, we believe these preliminary results are very encouraging and we will soon conduct new game sessions with domain experts.

5 Inserting Artificial Agents into the SimParc Game

We are currently inserting artificial agents into the prototype. We consider three types of artificial agents: the park manager, the artificial players and the assistants.

The park manager acts as an arbitrator in the game, making a final decision for types of conservation for each landscape unit and explains its decision to all players. He may be played by a human or by an artificial agent. We have implemented a prototype implementation of an artificial park manager, based on 2 steps: (1) internal/individual decision by the park manager, based on some argumentation model; (2) merging of the decision by the manager with the votes by the players, based on decision theory (social choice). Traces of argumentation may be used for explaining the rationale of the decision. The artificial park manager architecture is detailed in Section 6.

For the artificial players, we build up on previous experience on virtual players in a computer-supported role playing game, ViP-JogoMan [1]. The idea is to potentially replace some of the human players by artificial players (artificial agents). The two main motivations are: (1) the possible absence of sufficient number of human players for a game session and (2) the need for testing in a systematic way specific configurations of players' profiles. The artificial players will be developed along the artificial park manager existing architecture, with the addition of negotiation and interaction modules. We plan to use its argumentation capabilities to generate and control the negotiation process. In a next stage, we would like to explore the automated analysis of recorded traces of interaction between human players, in order to infer models of artificial players. In some previous work [5], genetic programming had been used as a technique to infer interaction models, but we also plan to explore alternative induction and machine learning techniques, e.g., inductive logic programming.

The last type of artificial agent is an intelligent assistant agent. This agent is designed to assist a player by performing two main tasks: (1) to help participants in playing the game, e.g.: the assistant agent tells the player when he should make decisions; what are the phases of the game; what should be done in each phase; etc.; (2) to help participants during the negotiations. For this second task, we would like to avoid intrusive support, which may interfere in his decision making cognitive process. We have identified some actions, e.g., to identify other players' roles with similar or dissimilar goals, which may help the human player to identify possible coalitions or conflicts. The general main idea for an assistance for negotiation is thus to combine and classify important information to help participants to make analysis and do it faster than they would do alone, while keeping their focus on the game proposal. An initial prototype implementation of an assistant agent has been implemented [7].

6 The Park Manager Artificial Agent

In this section, we describe an agent architecture to implement the park manager cognitive decision rationale. As we summarized before, our decision model is based on two mechanisms. These mechanisms could be viewed as modules of decision subprocesses. We believe that complex decision making is achievable by sequential organization of these modules. Before proceeding to the description of our agent architecture, we present some more detailed motivation for it.

6.1 Objectives

Participatory management aims to emphasize the role of local actors in managing protected areas. However, the park manager is the ultimate arbiter of all policy on devolved matters. He acts like an expert who decides on validity of collective concerted management policies. Moreover, he is not a completely fair and objective arbiter: he still brings his personal opinions and preferences in the debate. Therefore, we aim to develop an artificial agent modeling the following behaviors.

Personal preferential profile: The park manager decision-making process is supposed to be influenced by its sensibility to natural park stakes and conflicts. In decision theory terms, we can affirm that the park manager's preferential profile could be intended as a preference relation over conservation policies. One of the key issues is to understand that we cannot define a strict bijection between preferential profile and preference relation. The agent's preference relation is partially dependent on natural park resources and realities. Moreover, this relation is not likely to be an order or a preorder. Hence, our agent must be able to define dynamically its preference relation according with its preferential profile. We distinguish two preferential profiles:

- Preservationist: aims to preserve ecosystems and the natural environment.
- Socio-conservationist: generally accepts the notion of sustainable yield - that man can harvest some forest or animal products from a natural environment on a regular basis without compromising the long-health of the ecosystem.

Taking into account stakeholders' decisions: A participatory decision-making leader seeks to involve stakeholders in the process, rather than taking autocratic decisions. However, the question of how much influence the stakeholders are given may vary on the manager's preferences and beliefs. Hence, our objective is to model the whole spectrum of participation, from autocratic decisions to fully democratic ones. To do so, we want the park manager agent to generate a preference preorder over conservation policies. This is because it should be able to calculate the distance between any two conservation policies. This way, we can merge the stakeholders' preference preorders with the manager's one to establish one participatory final decision. The autocratic/democratic manager attitude will be modeled by an additional parameter during the merge process.

Expert decision: The park manager's final decision must consider legal constraints related to environmental management; otherwise, non-viable decisions would be presented to the players, thus invalidating game's learning objectives. These constraints are directly injected in the cognitive process of the agent. Hence, the agent will determine a dynamic preference preorder, according to its preferential profile, over allowed conservation levels.

Explaining final decision: In order to favor the learning cycle, the park manager agent must be able to explain its final decision to the players. We can consider that the players could eventually argue about its decision; the agent should then defend its purposes using some kind of argumentative reasoning. Even if such cases will be explored in future work, it is our concern to conceive a cognitive architecture which provides a good basis for managing these situations.

6.2 Architecture Overview

Let us now present an architecture overview of the park manager agent. As depicted at Figure 6, the agent's architecture is structured in two phases. We believe that sequential decision-making mechanisms can model complex cognitive behaviors along with enhanced explanation capabilities.

The first decision step concerns the agent's individual decision-making process: the agent deliberates about the types of conservation for each landscape unit. Broadly speaking, the park manager agent builds its preference preorder over allowed levels of conservation. An argumentation-based framework is implemented to support the decision making. The next step of our approach consists in taking into account the players' preferences. The result of the execution is the modified park manager decision, called the agent participatory decision, according to the stakeholder's preferences.

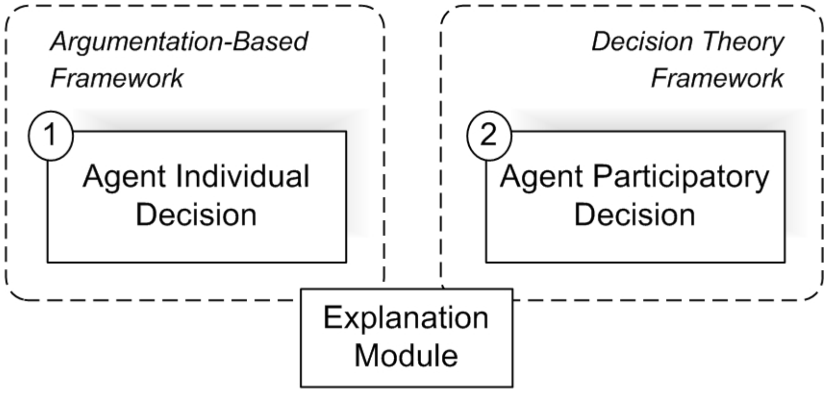

Argumentation-Based Framework
Decision Theory Framework
1
Agent Individual Decision
2
Agent Participatory Decision
Explanation Module

Fig. 6. Park manager agent 2-steps decision process

6.3 Agent Individual Decision

Recently, argumentation has been gaining increasing attention in the multi-agent community. Autonomous and social agents need to deliberate under complex preference policies, related to the environment in which they evolve. Generally, social interactions bring new information to the agents. Hence, preference policies need to be dynamic in order to take into account newly acquired knowledge. Dung's work [8] proposes formal proof that argumentation systems can handle epistemic reasoning under open-world assumptions, usually modeled by nonmonotonic logics. Argumentation thus becomes an established approach for reasoning with inconsistent knowledge, based on the construction and the interaction between arguments. Recently, some research has considered argumentation systems capabilities to model practical reasoning, aimed at reasoning about what to do [9,10,11]. It is worth noticing that argumentation can be used to select arguments that support available desires and intentions. Consistent knowledge can generate conflicting desires. An agent should evaluate pros and cons before pursuing any desire. Indeed, argumentative deliberation provides a mean for choosing or discarding a desire as an intention.

We could argue that open-world assumptions do not hold in our context. The agent's knowledge base is not updated during execution, since it is not directly exposed to social interactions. Knowledge base and inference rules consistency-checking methods are, therefore, not necessary. However, one key aspect here is to conceive an agent capable of explaining its policy making choices; our concern is to create favorable conditions for an effective and, thus closed, learning cycle. We believe that argumentation "tracking" represents an effective choice for accurate explanations. Conflicts between arguments are reported, following agent's reasoning cycle, thus enhancing user comprehension.

From this starting position, we have developed an artificial agent on the basis of Rahwan and Amgoud's work [10]. The key idea is to use an argumentation system to select the desires the agent is going to pursue: natural park stakes and dynamics are considered in order to define objectives for which to aim. Hence, decision-making process applies to actions, i.e. conservation levels, which best satisfy selected objectives. In order to deal with arguments and knowledge representation, we use first-order logic. Various inference rules were formulated with the objective of providing various types of reasoning capability. For example, a simple inference rule for generating desires from beliefs, i.e. natural park stakes, is:

$$\mathit{Fire} \rightarrow \mathit{Avoid\_Fires},\ 4$$

where *Fire* (fire danger in the park) is a belief in agents knowledge base and *Avoid_Fires* is the desire that is generated from the belief. The value 4 represents the intensity of the generated desire.

Examples of rules for selecting actions, i.e. level of conservation, from desires are:

$$\mathit{Primitive} \rightarrow \mathit{Avoid\_Fires},\ 0.4$$

$$\mathit{Intangible} \rightarrow \mathit{Avoid\_Fires},\ 0.8$$

where *Primitive* and *Intangible* represent the levels of conservation and 0.4 and 0.8 represent their utilities in order to satisfy the desire *Avoid_Fires.*

6.4 Agent Participatory Decision

Despite participatory ideals, a whole spectrum of park managers, from autocratic to fully democratic ones, can be measured, depending on how more participatory and democratic decision-making is operationalized. We propose a method, fitted into the social-choice framework, in which participatory attitude is a model parameter.

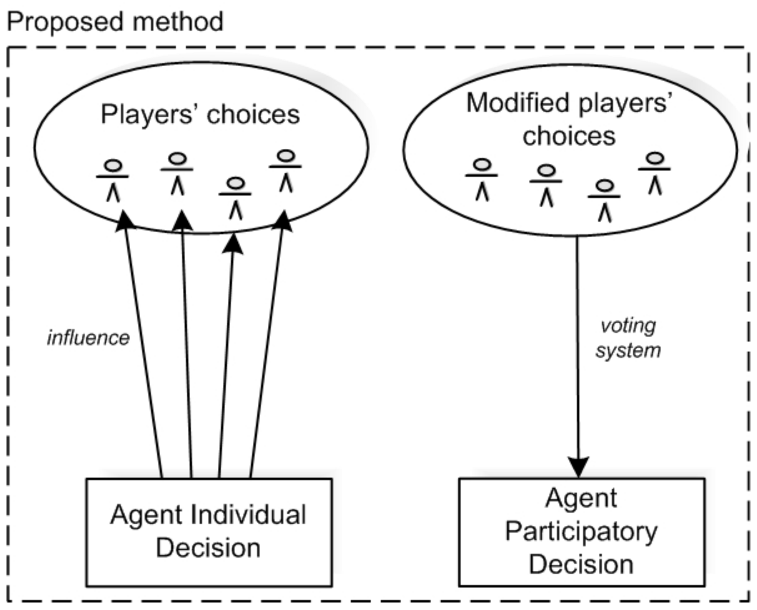

Proposed method
Players' choices
Modified players' choices
influence
voting system
Agent Individual Decision
Agent Participatory Decision

Fig. 7. Park manager agent participatory decision

In a real case scenario, a decision-maker would examine each stakeholder's preferences in order to reach the compromise that best reflects its participatory attitude. Our idea is to represent this behavior by weighting each player's vote according to the manager's point of view.

This concept is illustrated at Figure 7. The process is structured in two phases. Firstly, the manager agent injects its own preferences into the players' choices by means of an influence function describing the agent's participatory attitude. Stronger influence translates into more autocratic managers. Secondly, the modified players' choices are synthesized, using an aggregation function, i.e. Condorcet voting method. The result of the execution will be the agent participatory decision.

Example: Let the following be players' choices, where $\succ$ is a preference relation (a $\succ$ b means "a is preferred to b"):

$$player_1 = Intangible \succ Primitive \succ Extensive, \mathbf{v}_1 = (3, 2, 1)$$

$$player_2 = Extensive \succ Primitive \succ Intangible, \mathbf{v}_2 = (1, 2, 3)$$

$$player_3 = Primitive \succ Extensive \succ Intangible, \mathbf{v}_3 = (1, 3, 2)$$

Let Manager individual decision be:

$$manager_{ind} = Extensive \succ Primitive \succ Intangible, \mathbf{v}_M = (1, 2, 3)$$

The players' choices are converted into numeric vectors specifying the candidates' rank for each vote. Let the following be the influence function:

$$\odot(x,y) = \begin{cases} x & \text{if } x = y \\ x * 1/|x-y| & \text{otherwise} \end{cases}$$

The modified players' vectors will be:

$$\begin{aligned} \mathbf{mv}_1 &= \langle \odot(\mathbf{v}_1(1), \mathbf{v}_M(1)), \odot(\mathbf{v}_1(2), \mathbf{v}_M(2)), \odot(\mathbf{v}_1(3), \mathbf{v}_M(3))\rangle \\ &= (1.5, 2, 0.5) \\ \mathbf{mv}_2 &= (1, 2, 3) \\ \mathbf{mv}_3 &= (1, 3, 2) \end{aligned}$$

In order to find the manager participatory decision, we apply the Choquet integral $\mathcal{C}_\mu$ [12] choosing a symmetric capacity measure $\mu(S) = |S|^2/|\mathcal{A}|^2$, where $\mathcal{A}$ is the candidates set.

$$\mathcal{C}_\mu(\mathit{Intangible}) = 1.05,\ \mathcal{C}_\mu(\mathit{Primitive}) = 2.12,\ \mathcal{C}_\mu(\mathit{Extensive}) = 1.27$$

The result of the execution will then be:

$$\mathit{manager}_{\mathit{part}} = \mathit{Primitive} \succ \mathit{Extensive} \succ \mathit{Intangible}$$

Further details about the architecture formal background and implementation are reported in [13].

6.5 Implementation Framework

The architecture presented in this paper has been implemented in the Jason multi-agent programming language platform [14]. Besides interpreting the original AgentSpeak(L) language, thus disposing of logic programming capabilities, Jason also features extensibility by user-defined internal actions, written in Java. Hence, it has been possible to easily implement aggregation methods.

6.6 Examples of Results

The presented manager agent architecture and its first implementation have been tested over different scenarios. Simulations conducted in laboratory have been validated by team experts. However, more results from further tests are expected. We report hereafter an example of explanation for the managers decision over a landscape unit. Let the manager individual decision be the following:

$$\mathit{manager}_{\mathit{ind}} = \mathit{Intangible} \succ \mathit{Recuperation}$$

Arguments for *Intangible* are:

$$\mathit{Endangered\_species}\ \&\ \mathit{Tropical\_forest} \rightarrow \mathit{Maximal\_protection}$$

$$\mathit{Intangible} \rightarrow \mathit{Maximal\_protection}$$

Arguments for *Recuperation* are:

$$\mathit{Fire}\ \&\ \mathit{Agricultural\_activities} \rightarrow \mathit{Recover\_deteriorated\_zone}$$

$$\mathit{Recuperation} \rightarrow \mathit{Recover\_deteriorated\_zone}$$

7 Conclusion

In this paper, we have presented the SimParc project, a role-playing serious game aimed at computer-based support for participatory management of protected areas. The lack of human resources implied in RPG gaming process acts as a constraint to the fulfillment of pedagogical and epistemic objectives. In order to guarantee an effective learning cycle, park manager role must be played by a domain expert. Required expertise obviously narrows game's autonomy and limits its context of application. Our solution to this problem is to insert artificial agents into the game. In this paper, we focused on the architecture to implement park manager cognitive decision rationale. The decision-making agent can justify its behavior and generate a participatory decision. Our argumentation system is based on [9,10]: conflicts between arguments can be reported thus enhancing user comprehension. Moreover, we presented a decision theory framework responsible for generating a participatory decision. To the best of our knowledge and belief, this issue has not yet been adressed in the literature. Although more evaluation is needed, we believe the initial game session tests are encouraging. The final integration of the park manager agent within the new version of the prototype is under way. We plan new game sessions with experts in January 2010. Besides the project specific objectives, we also plan to study the possible generality of our prototype for other types of human-based social simulations. In the current architecture of the artificial park manager, only static information about the park and about the votes by players are considered. We are considering exploring how to introduce dynamicity in the decision model, taking into account the dynamics of negotiation among players (the evolution of decisions by players during negotiation).

Acknowledgments. We thank Diana Adamatti, Ivan Bursztyn, Paul Guyot, Altair Sancho and Davis Sansolo for their past participation to the project. We acknowledge current or past support from the following programs: the ARCUS Program (French Ministry of Foreign Affairs, Région Ile-de-France), the MCT/CNPq/CT-INFO Grandes Desafíos Program from (Brazil), and the CNRS-Cemagref Ingénierie Ecologique Program (France).

References

1. Adamatti, D., Sichman, J., Coelho, H.: Virtual players: From manual to semi-autonomous RPG. In: Barros, F., Frydman, C., Giambiasi, N., Ziegler, B. (eds.) AIS-CMS 2007 International Modeling and Simulation Multiconference (IMSM 2007), Buenos Aires, Argentina, pp. 159–164. The Society for Modeling Simulation International, SCS (February 2007)
2. Briot, J.P., Guyot, P., Irving, M.: Participatory simulation for collective management of protected areas for biodiversity conservation and social inclusion. In: Barros, F., Frydman, C., Giambiasi, N., Ziegler, B. (eds.) AIS-CMS 2007 International Modeling and Simulation Multiconference (IMSM 2007), Buenos Aires, Argentina, pp. 183–188. The Society for Modeling Simulation International, SCS (2007)

3. Irving, M.: Áreas Protegidas e Inclusão Social: Construindo Novos Significados. Aquarius, Rio de Janeiro (2006)
4. Barreteau, O.: The joint use of role-playing games and models regarding negotiation processes: Characterization of associations. Journal of Artificial Societies and Social Simulation 6(2) (2003)
5. Guyot, P., Honiden, S.: Agent-based participatory simulations: Merging multi-agent systems and role-playing games. Journal of Artificial Societies and Social Simulation 9(4) (2006)
6. Vasconcelos, E., Briot, J.P., Irving, M., Barbosa, S., Furtado, V.: A user interface to support dialogue and negotiation in participatory simulations. In: David, N., Sichman, J.S. (eds.) Multi-Agent-Based Simulation IX. LNCS (LNAI), vol. 5269, pp. 127–140. Springer, Heidelberg (2009)
7. Vasconcelos, E., Melo, G., Briot, J.P., Patto, V.S., Sordoni, A., Irving, M., Alvarez, I., Lucena, C.: A serious game for exploring and training in participatory management of national parks for biodiversity conservation: Design and experience. In: VIII Brazilian Symposium on Games and Digital Entertainment (SBGAMES 2009), Rio de Janeiro, Brazil (October 2009)
8. Dung, P.M.: On the acceptability of arguments and its fundamental role in non-monotonic reasoning, logic programming and n-person games. Artificial Intelligence 77(2), 321–357 (1995)
9. Moraitis, P.: Un modèle de raisonnement pour agents autonomes fondé sur l'argumentation. In: Journées Nationales de Modèles de Raisonnement (JNMR 2003), Paris, France (November 2003)
10. Rahwan, I., Amgoud, L.: An argumentation based approach for practical reasoning. In: 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), Hakodate, Japan, pp. 347–354. ACM, New York (2006)
11. Amgoud, L., Kaci, S.: On the generation of bipolar goals in argumentation-based negotiation. In: Rahwan, I., Moraïtis, P., Reed, C. (eds.) ArgMAS 2004. LNCS (LNAI), vol. 3366, pp. 192–207. Springer, Heidelberg (2005)
12. Choquet, G.: Theory of capacities. Annales de l'Institut Fourier 5, 131–295 (1953)
13. Sordoni, A.: Conception et implantation d'un agent artificiel dans le cadre du projet SimParc. Master's thesis, EDITE, Université Pierre et Marie Curie (Paris 6), Paris, France (September 2008)
14. Bordini, R.H., Wooldridge, M., Hübner, J.F.: Programming Multi-Agent Systems in AgentSpeak using Jason. Wiley Series in Agent Technology. John Wiley & Sons, Chichester (2007)

NonKin Village: An Embeddable Training Game Generator for Learning Cultural Terrain and Sustainable Counter-Insurgent Operations

Barry G. Silverman[1], David Pietrocola[1], Nathan Weyer[1], Ransom Weaver[1], Nouva Esomar[1], Robert Might[2], and Deepthi Chandrasekaran[1]

[1] Ackoff Collaboratory for Advancement of the Systems Approach (ACASA)
University of Pennsylvania, Philadelphia, PA 19104-6315
basil@seas.upenn.edu
[2] Innovative Management Concepts, Inc., Dulles, VA 20166

Abstract. This article describes a virtual village we are currently assembling. Called NonKin Village, this is a gameworld that brings life to factional agents in a sort of an emergent SimCity. It supports street level interaction and dialog with agents to learn their issues, needs, grievances, and alignments and to try to assist them in countering the agenda of an insurgent faction aimed at ending the rule of law. The player can attempt tactical Diplomatic, Information, Military, and Economic (DIME) actions and observe Political, Military, Economic, Social, Infrastructure, and Information (PMESII) effects unfold, but watch out! It's easy to go wrong in this foreign culture, to undertake operations with spurious side-effects. The player's goal is to learn enough about the foreign culture and their situation so as to be successful at influencing the world and facilitating a new dynamic to take effect, that of equitable and self-sustaining institutions.

Keywords: training game, cultural terrain, socio-cognitive agents, interactive environments.

1 Introduction

Immersive training environments have gained interest in a variety of fields and industries as a way to effectively teach skills with a large degree of flexible and cost-effective scenario authoring. Such flexibility, however, can lead to important design considerations for managing large and disparate collections of data in editing and run-time environments. From terrain maps to agent relationships, to cultural context, a great deal of information is required to rapidly generate an immersive environment. We have developed an architecture to support such an environment by facilitating content generation, customization, reuse, and plug-in functionality, as well as bridging capability with other technologies and visual platforms.

The use case for such an architecture is a training game, called NonKin Village, where the player(s) interacts with other virtual or real followers and leaders of contending factions at a local village level. These factions offer a corrupt SimCity-type of world where one must convince various "crime" families to convert to legit operation. NonKin is used to simulate insurgent operations in the village. The insurgent leader uses recruits to carry out missions. The player(s) has constrained resources, and must use them judiciously to try and influence the world via an array of Diplomatic, Intelligence, Military, and Economic (DIME) actions. The outcomes are presented as a set of intended and unintended Political, Military, Economic, Social, Informational, and Infrastructure (PMESII) effects.

The goal is to push the player through the three stages of counter-insurgency (COIN) theory: survey the social landscape, make friends/co-opt the agenda, and foster self-sustaining institutions so the player can safely depart [1]. The player learns to use the given resources judiciously and in a culturally sensitive way to achieve desired outcomes. All agents in the game are conversational and are able to explain their internal states, group grievances, relations/alignments, fears, and wants. The agents carry out daily life functions in the village in order to satisfy their internal needs for sleep, sustenance, company/belonging, maintaining relationships, prayer, etc. The village has places of employment, infrastructure, government and market institutions, and the leaders (agents) manage the economic and other institutional resources of their factions.

1.1 Theories of Insurgency/Terrorism

To begin, one can readily envision an insurgency existing in a world where a number of factions or clans range across the spectrum from those desiring the rule-of-law to those interested in chaos and regime change for any of a variety of reasons (ethno-political grievance, greed, crime, etc.). This is depicted across the top of Fig. 1. Indeed, in the Maoist theory of armed struggle, the preparatory stage of an insurgency is characterized by actions that seek to affect separate factions of the population of the nations or regions they are trying to influence, causing different factions to iterate (dynamically) through several states ranging from animosity and paralyzing fear to sympathy and membership in the insurgent movement [3].

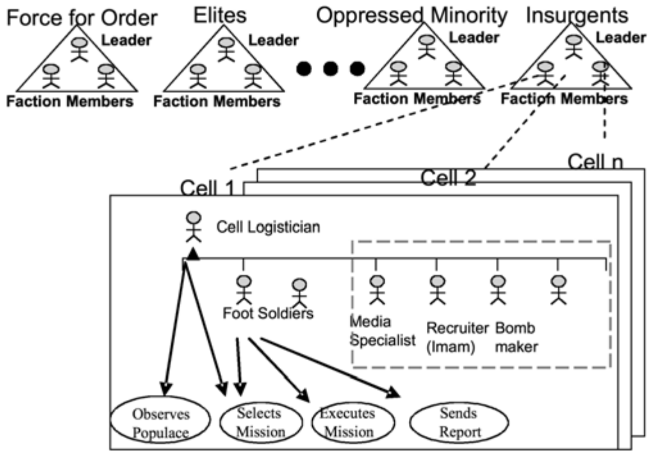


Fig. 1. Factions and agents in an insurgent area

Ideally one would like to realistically simulate such behaviors for the purposes of being able to train against it and analyze what influences it in a given area of operation. To train/analyze how to co-opt the agenda of an insurgency and mobilize the populace toward the rule of law one needs a set of simulated factions and insurgent agents readily adapted to any given region. Since members of a given populace will be at varying degrees of support for and participation with each side, this implies that the aim of counter-insurgency is not solely to destroy groups at the enemy end of the spectrum, but also to progressively shift individuals and groups closer to the friendly end. In fact, since insurgent cells are often hidden amongst supporting members of the population (bottom of Fig. 1), a focus strictly on reaction to insurgent attacks can be counter-productive. It will leave the agenda in their hands, cause collateral damage to potentially woo-able factions, and make the force for order seem to have no successful agenda of its own. Instead one must encourage the force for order to use a “full spectrum” of approaches to help diagnose the source of grievance, attempt to ameliorate the root causes, and build up whatever services and institutions that are lacking and potentially also causing discontent: e.g., see [1-3, 6-8, 13].

2 Overview of Socio-Cognitive Agents and Architecture

We have been assembling an architecture able to support the authoring of games in this milieu. This section overviews the main components of the architecture. Sun [12] and Zacharias et al. [14] provide a useful survey of the respective fields of social agents and cognitive agents and show that there are very few meso-level environments that straddle both topics to provide socio-cognitive architectures. Let us therefore, briefly review its major layers – social and cognitive.

2.1 Profiling Cognitive Agents

PMFserv is a COTS (Commercial off the Shelf) human behavior emulator that drives agents in simulated gameworlds. This software was developed over the past ten years at the University of Pennsylvania as a “model of models” architecture to synthesize many best available models and best practice theories of human behavior modeling [10]. PMFserv agents are unscripted, using their micro-decision making to react to actions as they unfold and to plan out responses. A performance moderator function (PMF) is a micro-model covering how human performance (e.g., perception, memory, or decision-making) might vary as a function of a single factor (e.g., event stress, time pressure, grievance, and so on.). PMFserv synthesizes dozens of best available PMFs within a unifying mind-body framework and thereby offers a family of models where micro-decisions lead to the emergence of macro-behaviors within an individual. For each agent, PMFserv operates its perception and runs its physiology and personality/value system to determine coping style, emotions and related stressors, grievances, tension buildup, impact of rumors and speech acts, and various mobilization and collective and individual action decisions to carry out the resulting and emergent behaviors. None of these PMFs are "home-grown"; instead they are culled from the literature of the behavioral sciences. Users can turn on or off different PMFs to focus on particular aspects of interest. When profiling an individual, various personality and cultural profiling instruments are utilized with GUI sliders and web interviews to elicit the parameter estimates from a country, leader, or area expert.

A key concept in PMFserv that assists in modular modeling and object reuse is the implementation of Gibson's affordance theory to manage when and how agents and objects may be perceived and acted on. Each entity in the world applies perception rules to determine how it should be perceived by each perceiving agent. Objects then reveal the actions (and the potential results of performing those actions) afforded to the agent. For example, an object representing a car might afford a driving action which can result in moving from one location to another. A business might afford running it, working there, purchasing goods, and/or attacking and damaging it. These affordance markups permit PMFserv agents to perceive and reason about the world around them.

2.2 FactionSim – The Social System Layer

FactionSim is an environment that captures a globally recurring socio-cultural "game" that focuses upon inter-group competition for control of resources (e.g, Security/Economic/Political Tanks) [11]. It is a tool that allows conflict scenarios to be established in which the factional leader and follower agents all run autonomously and are free to employ their micro-decision making as the situation requires. This environment facilitates the codification of alternative theories of factional interaction and the evaluation of policy alternatives. A faction has roles for one or more leaders; various ministers influenced by the leaders and in charge of institutions that tap the resources and provide services; and several named and/or archetypical sets of followers (e.g., core, fringe, and hierarchies of sub-groups). FactionSim uses PMFserv to fill these leader-minister-follower roles. Factions can be clustered into super-groups ahead of time. They also may dynamically form or break alliances by autonomous agent decision making. Analysts can interact with the FactionSim and attempt to employ a set of DIME actions to influence outcomes and PMESII effects. FactionSim has recently been extended to handle specific follower groups (military, bureaucracy, elites, religious groups, etc.) and a module was added that contains economic institutions that manage the allocation and distribution of public and some private goods. It also supports various types of economies (i.e., tribal, developmental, black market, etc.) and various regime types ranging from authoritarian to democracies with election processes. Third party economic models can also be used to supplant this set.

2.3 VillageSim-FactionSim Interactions

These two simulations – FactionSim and VillageSim, a PMFserv scenario of a village – are used together in NonKin Village for investigating complex social environments that require multi-resolution models. Fig. 2 illustrates the concept, where FactionSim's scope focuses on macro models at leader, group, and institution levels that consider distribution of services, relative power between factions, and other societal mechanisms that reverberate down to an individual level. Each group leader corresponds to the same agent in the village layer, but the leader's action set is clearly different. Followers, on the other hand, represent several agents in the village and

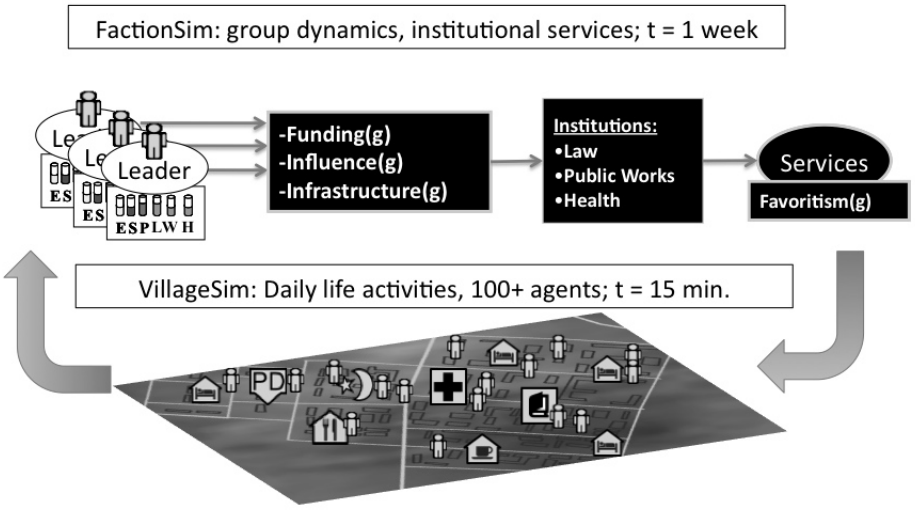

FactionSim: group dynamics, institutional services; t = 1 week
Leader
ESPLWH
-Funding(g)
-Influence(g)
-Infrastructure(g)
Institutions:
•Law
•Public Works
•Health
Services
Favoritism(g)
VillageSim: Daily life activities, 100+ agents; t = 15 min.
PD

Fig. 2. Multi-resolution simulation architecture

their decisions center around support and opposition of groups. On the VillageSim side, hundreds of agents go about daily life routines and detailed interactions and dialogues between these villagers and players are possible.

The multi-resolution, dual simulation setup can also facilitate investigations in emergent behavior. For example, a high-level policy decision made by a leader agent or a player in FactionSim can cause second and third-order effects in the village, or macro-to-micro causation. Updates in the economy model allocate capital to proxy agents, which then get distributed to those represented agents according to their individual socio-economic status. Conversely, consider player interactions with villagers in real time. Once the specific interactions are completed, the player will observe their overall impact either in 15 minute game time intervals if VillageSim is controlling the clock, or in a game week if FactionSim controls it.

2.4 World Objects

World objects are all physical instances that afford actions to agents. Structures, stores, homes and symbolic items are just a small collection of object categories. For the purposes of the architecture in providing a functional framework for immersive training environments, several taxonomies have been developed to classify object types along with their functional affordances (further discussion is outside the scope of this paper, but is addressed in [11]). In NonKin a great many objects are pre-encoded in its libraries using these taxonomies so that training scenario developers need not fill in the markups, but only need to link them naturally to structures, areas, organizations, etc. of that town or region.

2.5 Cultural Context

Since we are modeling complex social environments with an assumed heterogeneity of cultures between agents or players and agents, it is critical to include cultural information that may affect decision making and perception. These cultural cues are packaged in the gameworld as abstract objects and they are represented in the same way as concrete world objects. Indeed, we make use of this representational form to store historical grievances and new social transgressions as they occur in the simulation. By "social transgression" we mean an offense an agent can commit against social rules. We utilize a simple, yet comprehensive taxonomy of the types of transgressions possible [5]. All transgressions have a transgressor, a set of victims, and a set of effects. Effects are the direct effects of the offending action, not the emotional effects on observers. Those events or effects are handled internally by the PMFserv agents as described in [10]. Beyond these basic properties, our transgression objects keep track of some relations with the transgressor, relations with observers, properties of the effects, and relations between the transgressor and observers. The transgression objects also keep track of atonement actions emanating from transgressors and their agents (e.g., compensation, apology, etc.) and forgiveness actions by the victims.

A major focus of NonKin Village is to provide players with immersive and accurate interactions with culturally and socially competent agents. To facilitate this, another abstract object called an interaction object is introduced into the world. These objects provide a context based on defined cultural and social rules, or norms. From mundane daily interactions such as buying coffee, to complex discussions involved in business negotiations or a dowry, agents assume roles governed by the class of interaction in which they join at a particular moment. Roles include a relevant action set as well as information on how that role should be perceived by others in the interaction. These data are part of the authoring process for the developer while the implementation is handled by the architecture.

2.6 The Architecture

NonKin's architecture needed to address several general problems. Primarily it needed a clean way to integrate multiple data sources and engines together in a modular way while still keeping reasonable load times. This was done by breaking NonKin into two basic pieces, AESOP and NonKin VillageGame.

Fig. 3 illustrates a general breakdown of the major pieces involved. AESOP (Authoring Electronic Stories for Online Players) is responsible for taking the various disconnected pieces of data and "baking" them into a single PMFServ library. This library is then pushed into a Data Layer which is responsible for presenting the library and its contained data to the NonKin VillageGame executable. Inside NonKin VillageGame we have an Interaction Engine that builds context-sensitive simulations for agents to operate within, and an Explanation Engine which exposes the various models in a conversational manner to the player and other agents.

NonKin VillageGame is accessed from the outside via a Bridge which exposes both the PMFServ Framework in general and the User Interface in particular. The User Interface sits inside the simulation and interacts with the world through the same model interface as socio-cognitive agents.

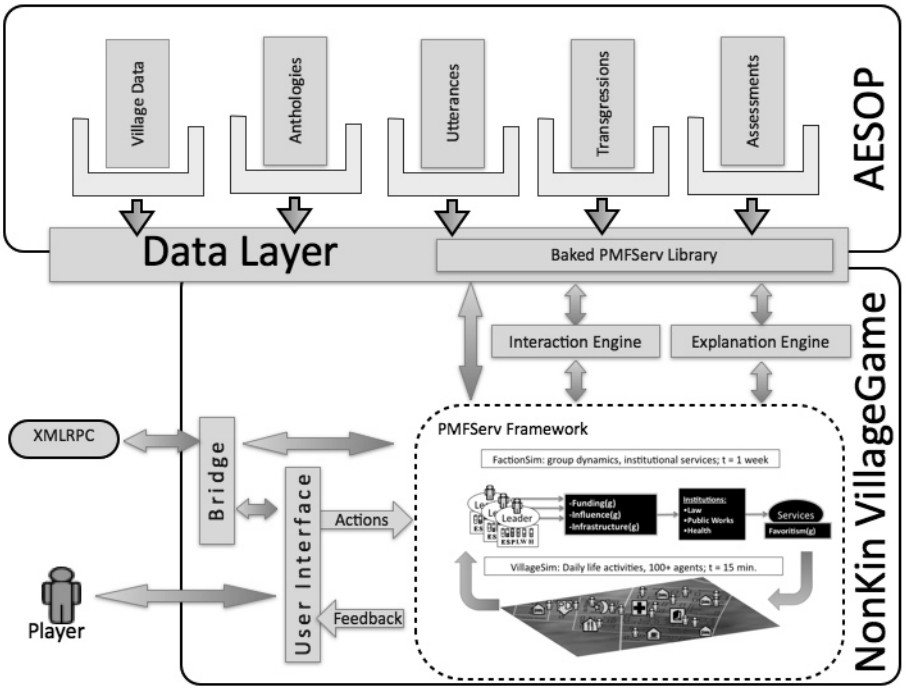


Fig. 3. NonKin architecture overview

2.6.1 Data Layer and the NonKinData Object

The Data Layer is implemented via a “NonKinData Object”, which exists as a singleton in global space and serves as the point of access for both running and editing villages. NonKinData is responsible for loading either the human readable component files or the single baked data file depending on what mode we are running in. As described in Section 3, it takes a set of AESOP libraries as inputs and produces a single baked PMFServ library which can be loaded into any PMFServ compliant environment and examined.

2.6.2 Components

The NonKinData object is also responsible for wrapping up (baking) various data components (as shown in the top row of Fig. 3) and providing a single point of access for asking questions about the village. Each individual component is stored in an easily edited intermediate format (XML) until the final step that encodes the pieces into a single PMFServ library. The components are described in further detail in Section 3.2.

3 AESOP and Conversational Abilities

A feature worth mentioning about PMFserv is that the agents are conversational. That is, a player may interrogate them about the parameter settings of any of their tanks, goals, standards, preferences, social relations, group dynamics, decision making history, opinions about other agents and the actions they have done, opinions about FactionSim institutions and organizations, how they feel about transgressions and transgressors, whether atonement is possible and how forgivable is the grievance, etc. PMFserv includes a narrative engine able to tell stories about what an agent knows [9]. In this way, the PMFserv agents can initiate dialogs when they want something from the player and they can give qualitative explanations when they talk. In terms of explanation, PMFserv agents do not parse natural language queries, but instead expose a large list of things you can talk to them about. These lists are dynamic and hierarchical so that the conversations are multi-stepped. Further, the agents include a "familiarity" scale (1-6) so they will reveal more things they are willing to discuss with the player the more familiar and trustworthy the player becomes to them. In terms of trustworthiness, the agents apply their full set of cognitions, emotions, social relation PMFs, etc., to the player and the player's group. So this means that they will grow closer to the player the more the player does good deeds for them. A player can do good deeds in the village in general, or via a structured dialog language make "social contracts" with specific agents that spawn objects the agents keep track of. Since PMFserv agents keep track of commitments and promises the player makes to them, it is equally possible to build trust with them and/or to violate expectations and cause their enmity.

3.1 AESOP and Village Editor

Given the number of different components that work together in NonKin to produce engaging interactions and conversations, it was necessary to solve the problem of how to expose these capabilities to knowledge engineers and training developers without making the process excessively complex or manual. The first version of the narrative editor depended heavily on user edited text files and arbitrary text fields that needed to match up between components. This method of editing tended to be labor intensive and error prone. It also made it increasingly difficult to add new features or change data file formats.

The solution was to enhance the AESOP editor, both in expanding its existing capabilities and giving it a new role in the authoring process. The AESOP editor serves two basic roles. Firstly it is responsible for coordinating the data of the various components and exposing them as an editor-friendly GUI. Its second task is to handle the actual baking of the various components into the final high-speed data file.

3.1.1 Storage

Like NonKin proper, AESOP has access to a single data object, NonKinData. NonKinData is responsible for tracking the various configuration files and their associated components. It is also responsible for providing an interface that allows components to access data without needing to be aware if it is working with a baked file or independent component data files.

3.2 Component Plug-Ins

Within the context of AESOP, a component refers to a package of related information that is capable of describing how to perform basic functions on itself like load, save, bake, and edit. Each component is responsible for handling its own data files, exposing its editing panel to the AESOP framework, and for providing a 'bake' function that encodes the component into the final PMFServ file.

By treating the component as plug-ins we give AESOP the ability to work with various village elements in a neutral way. AESOP does not need to know how a transgression works or what goes in to making an anthology. All it needs to be aware of is that there is a transgression and anthology component and display that information to the author. This allows us to add or remove significant functionality from a village without having to alter the core AESOP executable.

In Fig. 3 we saw a list of the major components we use to construct a NonKin Village:

Village Data - Village Data contains the bulk of the simulation elements in the village. It is broken up into two major pieces, the "library" and the "scenario". The library contains "entities", which are various agent and object archetypes used in the creation of specific instances of the village. For instance, the library might contain entries such as "Housewife" or "Sheik". The scenario on the other hand contains simulation-read "instances" of various agents. Instances are complete agents filled with the personal information that makes them individuals. For example, "Aridha Majid Mukhtar Zakar", a Shumar-group Housewife, would be an instance in the scenario.

Anthologies - Anthologies represent packages of states that form a single playable situation within the village. An anthology tracks which village we are using, what scenario implements the situation, which agents and objects are present, and what conversational elements are available to them. Anthologies also contain a number of possible vignettes, each of which is a self-contained conversational fragment that agents can access for conveying information to each other.

Utterances - The utterance catalog contains a list of culturally relevant statements that serve to bridge the high-level conversational capabilities of agents with the low-level models that implement their various aspects. Each utterance in the catalog represents a single line of conversation with markups describing how to pull information from underlying models and what information was learned by the listener.

Transgressions - The transgression catalog contains a list of culturally relevant transgressions that can potentially exist in the village. These transgression templates are baked into the library section of the Village Data and are instantiated into the scenario as transgressions occur. The transgression catalog is also responsible for storing grievances, which are pre-existing historical transgressions that have already occurred and thus are instantiated straight into the scenario at bake time for immediate usage.

Assessments - Assessments allow the author to explicitly define the criteria for successes and failures in a particular training scenario by mapping events and changes in properties to custom grading schemes. Assessments help answer the question of 'what was the outcome we wanted?' from a particular set of actions and results.

3.3 Baking

The bake process combines the components into a single PMFServ library file. It does this by first creating a blank PMFServ file and then iterating over the loaded components, asking each one to “bake” itself into the newly created file. Finally it saves this file out as a binary data block (rather then human readable text) in order to make it load more efficiently into a NonKinData object.

3.3.1 Entity Construction

Historically, most objects are defined explicitly in the village data file. When one is authoring an archetype they need to spell out every element every time. Related entities that share the bulk of their structure need to be defined each time with the small variations being inserted manually. If a meta-archetype needs to change, these changes must be performed manually to each archetype based off the larger class. This creates a situation where authoring becomes labor intensive and error prone, highly discouraging the altering of meta-archetypes.

The solution, outlined in Fig. 4, involves stepping back from PMFServ’s “entity-to-instance” relationship. Instead, we create a “proto-entity” within AESOP which is split into two separate objects during editing and combined during the bake process. This separation of template from metadata allows AESOP to build a catalog of reusable object archetypes that contain complex data like afforded actions and have simpler objects that contain only the information that makes each object unique.

An example of this would be a transgression template. An earlier implementation involved filling the PMFServ library with a collection of transgression entities. Each entity represented an archetype that was identical except for a small number of state variables which defined what made the individual template a specific type of transgression. If an author wanted to make any changes to what actions were afforded to transgressions in general, each transgression had to be changed (and tested) manually.

Under the Bake system however, we can have a single transgression template object with all the elements that make up a transgression except the elements that differentiate one type of transgression from another. These differences are instead stored in smaller metadata objects that allow for editing of just that subset of elements. During the bake process these two pieces are combined to create a collection of library entities similar to the ones PMFServ was accustomed to already

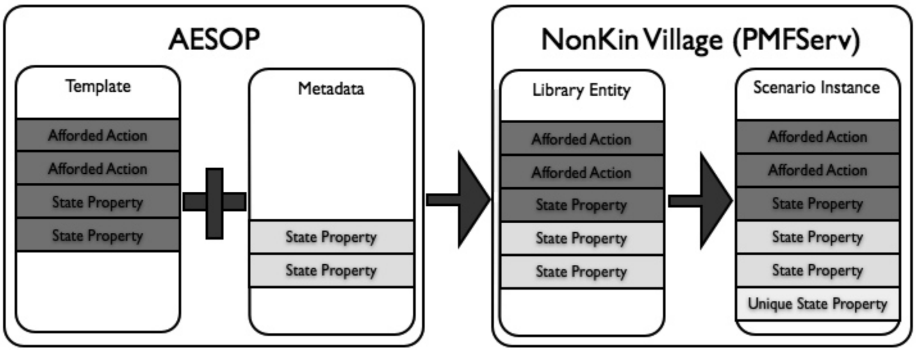


Fig. 4. Flow of object construction from template/metadata to final scenario instance

having. These entities can then be instantiated into individual scenarios as they are needed. This gives us a smooth scale of "general" to "specific", exposing the editable characteristics at each stage.

4 NonKin Overview

FactionSim is abstracted from geo-political considerations and, further, it makes no commitment to populating the factional leaders and followers in any sense in real locations, occupations, or other roles. We have a country sim generator that does this for state and international actor modeling. NonKin is the name of our generator intended to bring FactionSim into focus for human terrain in tactical regions. Specifically, NonKin is a region scenario generator meant for use to implement villages, towns, and city neighborhoods, including connectivity of these areas to higher level institutions and assets. It is a SimCity genre of game engine. It is a portable/plug-in and role playing game generator that is being outfitted with a reusable and extensible library of mock characters, institutions, factions, militias, and so on who carry out daily life and various economic, political, familial, and security activities in a culturally accurate way. Factions and institutions/organizations and roles are defined atop FactionSim, while agents are driven by the PMFserv engine.

A test scenario is currently being implemented for the United States Marine Corps (USMC). This is the hypothetical town of Hamariyah occupied by 200 individuals. The USMC staff from their 29 Palms training facility generated Hamariyah and descriptions of the town history, its 200 residents, 7 tribal groups, families, jobs, institutions, inter-factional grievances, and so on. This is a paper-based description, though some of it was provided in csv files that we recast into spreadsheet workbooks that were then read by the PMFserv model constructor. It is a plan for a mock town of role players that they have since deployed at 29 Palms.

The high level goals of NonKin include:

- Easy to use scenario generator -- profile actual personalities, cultures, group norms, historical grievances, institutions, infrastructure, etc. (based on turning on/off toggles from libraries of reusable agents, organizations, and other types of objects)
- Easy to use training content generator that offers a library of default training sets for the tasks, tactics, and procedures relevant to each of the three phases of the COIN theory as espoused in Military Field Manual (FM) 3-24 (See [13] but NOTE: since the Tactics, Techniques, and Procedures (TTPs) are unspecified by USMC doctrine, the defaults are examples taken from interviewing the 29 Palms observer/controllers or Coyotes. The default cases are editable and extensible).
- Conduct security operations, patrols, checkpoints, etc.
- Interact in different areas of town, in order to inventory and befriend the population. Discover who lives where and the residents' grievances, hopes, and stabilization and reconstruction needs. Also, find out intelligence about bad guys.
- Visit often, become familiar to the residents, observe the residents going about their daily routines, and learn to detect anomalous and suspicious behaviors before bad things actually happen.
- Make reconstruction commitments and further attempt to befriend factions and co-opt the agenda away from the simulated insurgent faction's leaders and followers.

Try out interventions of the Diplomatic, Information, Military, Economic, Financial, Intelligence, and Law Enforcement (DIME-FIL) type and observe PMESII effects and their spread through communities.
- Attempt to understand what is needed community-wide and in the institutional organizations that provide services and resources so that a self-sustaining peace can take sway and withdrawal is possible. Try to run some transition and NonKin agent training so the local leaders and followers can assume running of their own services.
- Commit mistakes and errors in this simulated world so you do not need to commit them in the real world that it simulates. Encounter multiple cases and situations for the TTPs of each of the COIN phases.
- Receive missions, in-game coaching, and after-action reviews from PMFserv-profiled agents.

4.1 Types of Training Available

We have spent nearly two years building up these diverse capabilities and have working prototypes of many of them. The first year was devoted to COIN phase I (inventory the human and cultural terrain), while the current year is aimed at COIN phase II (co-opt the agenda, make friends in the village). A series of video clips demo these capabilities.

The training is meant to unfold in three tiers that we are adding to constantly as well:

- Casual Interaction & Cultural Flashcards – This is a short and simple method of interacting with a given villager in a given context. It is useful for learning cultural norms about interacting with women, children, head of household, etc. Skills learned here will help in Lane training.
- Interaction-Oriented Lanes – Lanes are specific tasks such a searching a house, staffing a checkpoint, detaining a suspect, and/or talking to a tribal leader, among others. They are isolated tasks and one gets to rehearse and hear feedback on each of them separately.
- Grievance-Oriented FEX Game – This is a full village scenario. If the player does nothing, the factional leader and follower agents use their micro-decision making to act on their rivalries, grievances, and resource control concerns. If you mismanage the situation, various factions and members might be drawn to the insurgent faction's side. Alternatively, through the mechanics of the Interaction and Explanation Engines, the user should instead be able to dialog and interact with each character to try and influence the world so that a new dynamic takes effect, that of cooperation. Attempt tactical DIME actions and observe PMESII effects unfold.

In the next section we show some of the Flashcards to illustrate how trainees interact with NonKin Village.

4.2 Marine Training Flashcards

Due to space restrictions we can only describe the first of these in the current paper. Specficially, "flashcards" are a simplified, functioning demonstration game, and the purpose of each is to train skills and to introduce the player to how NonKin aims to train cultural skills. NonKin differs markedly from shooter games in that successful intercultural interactions is a primary means of achieving the game objective (i.e. stabilizing the village and banishing insurgency) rather than physically destroying an enemy, solving puzzles, etc. This is a novel approach and needs some introduction for the uninitiated player.

Many computer games begin with a short "boot camp" version of the game that orient the player to the user interface, player movement, game-play strategy, and the use of the tools or weapons found in the game. To achieve this in NonKin Village we needed to orient the player to the means and method of interacting with the synthetic agents and how this interaction affects the state of the village. What was needed was a "sandbox" or school where the player was instructed on the possibilities contained in NonKin. In the real world the USMC does this in the major pre-deployment exercise "Mojave Viper" at 29 Palms, CA. There the units practice population-centric counterinsurgency techniques by entering into focused role-playing exercises with non-English speakers in a mock village. Exercises start small, with restricted scope ("Lane" training) and expand to finally include the entire force in one large exercise ("FEX").

Flashcards introduce intercultural interactions by isolating the personal interaction and having a "trainer" persona introduce the situation and give feedback on the player's actions. Flashcards is composed of 6 vignettes dealing with some useful examples of interacting with persons in the conduct of a mission:

- Greetings and introductions
- Honor and property
- Women
- Children
- Detaining a person
- Rapport building

Note that nothing about these topics is "built-in" to the AESOP editor or NonKin Village. The person using AESOP to develop training scenarios has full discretion as to what content to put in place. These examples were gleaned from the extensive materials provided by USMC Center for Advance Operational Culture Learning (CAOCL) and from the USMC Mojave Viper role-playing scenarios [15].

4.2.1 Authoring the Flashcard Vignettes with AESOP

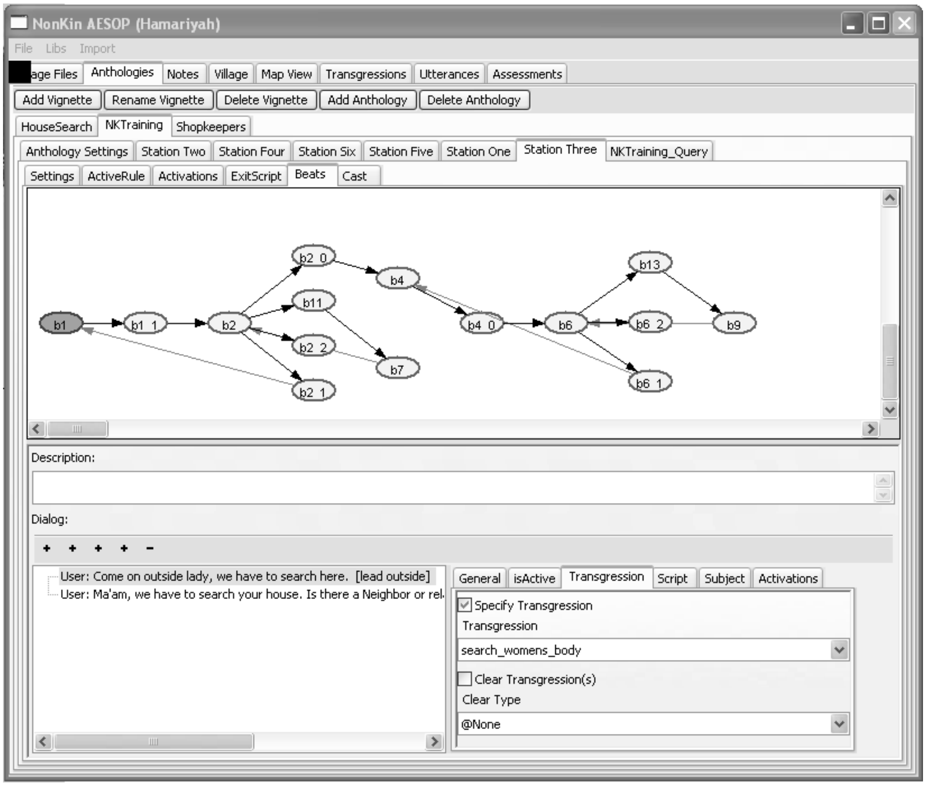
Each of the above vignettes is represented on the Hamariyah village map as a test station and the player receives instructions that they should visit each station for an orientation on using NonKin Village. Within each vignette the structure is the same:

- Trainer introduces a Hamariyah villager who will assist in this training exercise.
- Trainer indicates that after the interaction the player may query the villager on their feelings.
- Trainer introduces the situation.
- Player is presented with several statements they may make to the villager.
- After the villager processes any transgressions that might be associated with the statement and reacts the trainer gives a critique of the interaction.

Finally the player is given the following set of statements they may select from:

1. To trainer: I'd like to try another situation.
2. To trainer: I'd like to try this situation again.
3. To trainer: I think I'm finished with this station.
4. To villager: How do you feel about my actions?
5. To villager: May I ask you a question?

The agent's response is non-deterministic; it depends on the agent's current emotional state and their goals, standards and preferences (personality and values) as represented in its PMFServ model. The agent perceives the transgression and adapts its model accordingly. Then when the agent is presented with a choice of statements to make to the player, the agent examines the potential personal effects of saying each statement and decides which is most beneficial toward improving its emotional state. Below are two figures showing the AESOP editor being used to configure this process. While the top of these figures show the tabs for authoring each of the NonKinData components mentioned earlier, to limit the length of this chapter we will only describe using AESOP in the authoring of a single flashcard, that of interacting with women.


NonKin AESOP (Hamariyah)
File Libs Import
age Files Anthologies Notes Village Map View Transgressions Utterances Assessments
Add Vignette Rename Vignette Delete Vignette Add Anthology Delete Anthology
HouseSearch NKTraining Shopkeepers
Anthology Settings Station Two Station Four Station Six Station Five Station One Station Three NKTraining_Query
Settings ActiveRule Activations ExitScript Beats Cast
b1
b1 1
b2
b2 0
b11
b2 2
b2 1
b4
b7
b4 0
b6
b13
b6 2
b6 1
b9
General isActive Transgression Script Subject Activations

Description:

Dialog:

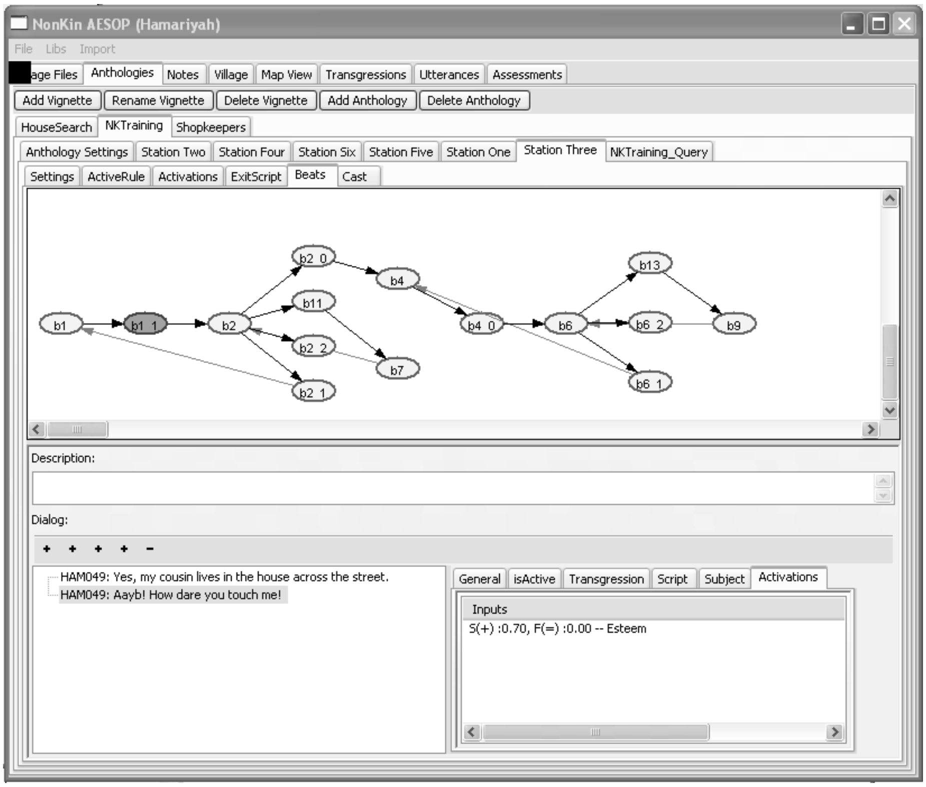
User: Come on outside lady, we have to search here. [lead outside]
User: Ma'am, we have to search your house. Is there a Neighbor or rel

☑ Specify Transgression
Transgression: search_womens_body
☐ Clear Transgression(s)
Clear Type: @None

Fig. 5. User statement with transgression

Fig. 5 shows beat 1 of the “Interacting with Women” Flashcard vignette, in which the player has two dialog choices. The selected option is marked up with a transgression. The other option has no transgression. Fig. 6 shows beat 1-1 with the agent’s possible responses, each of which is marked with an activation (utility of this particular response applied to values and personality nodes). The prior transgression, if it occurred, would have had a negative effect on the agent’s esteem. The agent would therefore process this selected response option (which will positively affect the agent’s esteem) as most beneficial to the agent’s emotional state. “Esteem” is a parameter from a pull-down menu exposing internal models that the agent reasons about.

After the agent reacts, the statements to the trainer allow the player (in beat b2) to access another interaction in the same vignette, retry the interaction, or leave to try another station. The statements directed to the villager allow the villager to express its inner state to the player via NonKin's Explanation Engine. One choice gives direct access to the agent's feelings about the player and may return, for example, “I disapprove of Squad Leader's actions”. The other allows access to a special query vignette where the player has an array of questions to pose to the agent. Some examples are: “Tell me about your family”, “Tell me about yourself”, “Tell me about the groups you know”. To the latter the agent would respond, “which one?” and the player would select a group, e.g. the Shumar tribe in Hamariyah Village. Then the agent might respond, “I distrust the Shumar”. Using the Explanation Engine to access the


NonKin AESOP (Hamariyah)
File Libs Import
age Files Anthologies Notes Village Map View Transgressions Utterances Assessments
Add Vignette Rename Vignette Delete Vignette Add Anthology Delete Anthology
HouseSearch NKTraining Shopkeepers
Anthology Settings Station Two Station Four Station Six Station Five Station One Station Three NKTraining_Query
Settings ActiveRule Activations ExitScript Beats Cast
b1 b1 1 b2 b2 0 b11 b2 2 b2 1 b4 b7 b4 0 b6 b13 b6 2 b6 1 b9
Description:
Dialog:
General isActive Transgression Script Subject Activations
Inputs

HAM049: Yes, my cousin lives in the house across the street.
HAM049: Aayb! How dare you touch me!

S(+) :0.70, F(=) :0.00 -- Esteem

Fig. 6. Agent response with activation affordances

agent's internal representation of self and surroundings is of limited utility in the scope of these simple flashcards, but having it included introduces the concept to the player.

As we see from this example, player actions (in this case, narrative statements) make an impression on the agent by way of transgressions attached to the player actions. Transgressions can be set on a 10-point scale over the unit interval along 4 separate axes of Faux Pas, Violence, Taboo and Materialistic transgression. Transgressions are intended, over time and with varying intensity, to change the agent’s perception of the player, the player’s group, and other agents and objects associated with the player/player group. Flashcards does not have enough scope of play for this to really happen, but the transgressions are available to demonstrate post-vignette assessment of the training of the player (see Section 4.3).

4.3 Illustrative Vignette

After the NonKin Data Layer bakes the flashcard into the library, this section shows what the player interaction with it looks like. Here is a listing of the dialog that appears screen by screen in the village dialog window. Figure 7 shows a piece of this in context.

<u>Vignette</u>: Interacting with Women

<u>Initial Trainer statement</u>: Marine, at this station we are practicing interacting with women. The objective is to understand what is appropriate and inappropriate conduct in several common situations. This is Aridha Majid Mukhtar Zakar, middle-aged resident of Hamariyah, member of the Shumar tribe, and housewife. She will assist you in this exercise. Feel free to make mistakes. She may be offended, but that's her job in this exercise. To begin you need to search her house.

Player statement options:

1. Ma'am, please come outside so we can search your house.
2. Come on outside lady, we have to search here. [lead outside]
3. Ma'am, we have to search your house. Is there a Neighbor or relative nearby who can sit with you while we conduct the search?

If player chooses statement 2, a transgression with the following intensity values is instantiated:

– Faux pas: 0.25
– Violence: 0.25
– Taboo: 0.75
– Materialistic: 0

<u>Agent response</u>: Aayb[shame on you]! How dare you touch me!

<u>Trainer critique</u>: You have greatly offended this woman by touching her and taking her out of her home. Her relatives will be very upset as well. Go back and try it differently or try a different situation.

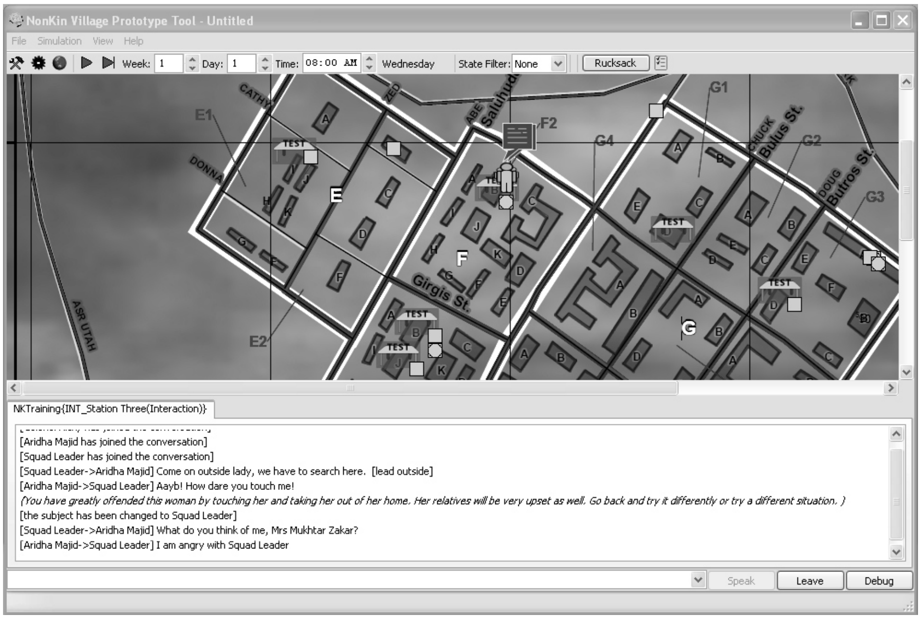
<u>Player queries Agent</u>: “What do you think of me, Mrs. Mukhtar Zakar?

<u>Agent response</u>: “I am angry with SquadLeader”.

Note that due to how we authored this in Section 4.1, the agent has to choose her response based on the emotional affordance of the statement alternatives (see Fig. 6).

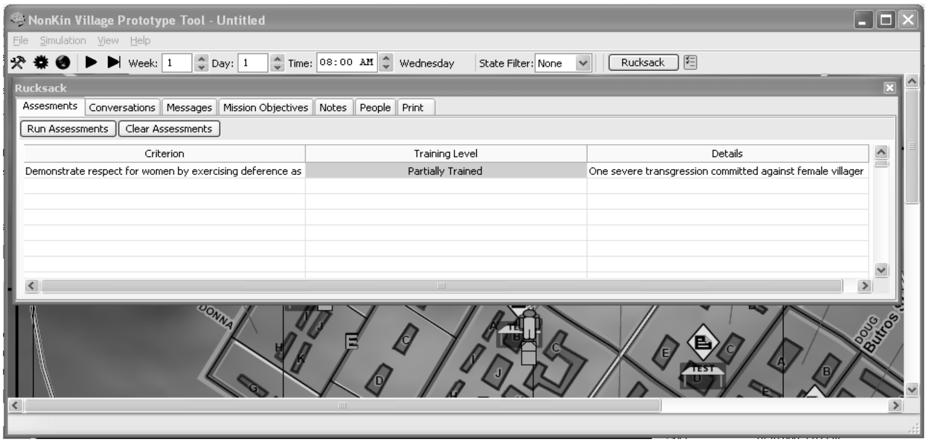
Her choice was not scripted or hardwired to follow a link to another node of a finite state machine. If her emotional model or value priorities were different, this affordance might not be so significant to her and her choice might be different.

Having now completed one cycle of the interaction, the player may now review an assessment of the interaction by exiting the vignette, selecting the assessments tab in the "Rucksack" and clicking "Run Assessments" (Fig. 8). The player will


NonKin Village Prototype Tool - Untitled
File Simulation View Help
Week: 1 Day: 1 Time: 08:00 AM Wednesday State Filter: None Rucksack
NKTraining{INT_Station Three(Interaction)}
Speak Leave Debug

```
[Aridha Majid has joined the conversation]
[Squad Leader has joined the conversation]
[Squad Leader->Aridha Majid] Come on outside lady, we have to search here.  [lead outside]
[Aridha Majid->Squad Leader] Aayb! How dare you touch me!
(You have greatly offended this woman by touching her and taking her out of her home. Her relatives will be very upset as well. Go back and try it differently or try a different situation. )
[the subject has been changed to Squad Leader]
[Squad Leader->Aridha Majid] What do you think of me, Mrs Mukhtar Zakar?
[Aridha Majid->Squad Leader] I am angry with Squad Leader
```

Fig. 7. Illustrative vignette in NonKin Village game


NonKin Village Prototype Tool - Untitled
File Simulation View Help
Week: 1 Day: 1 Time: 08:00 AM Wednesday State Filter: None Rucksack
Rucksack
Assesments Conversations Messages Mission Objectives Notes People Print
Run Assessments Clear Assessments

Criterion	Training Level	Details
Demonstrate respect for women by exercising deference as	Partially Trained	One severe transgression committed against female villager

Fig. 8. Rucksack assessments panel

see “Partially Trained” for CAOCL TCAT 005: “Professionalism (demonstrate professional behavior when interacting with locals, mindful of status, perspective and perceptions of the people with whom they interact and are not condescending or demeaning; demonstrate respect for women by exercising deference as much as the situation allows; seek permission to enter homes when it is practical to do so).”

4.4 Rucksack Tool

The rucksack UI is the place where the player stores collected information (orders, intel), reviews agent/player interactions, sees analysis of groups and events, and reviews after-action reports (assessments). Assessments may be configured on a per-vignette basis using specific training assessment metrics as defined by CAOCL. Areas such as rapport building, leader engagement, communication, and professionalism are assessed as “trained”, “partially trained” and “ineffective”. The player’s path through the vignette and transgressions accrued along this path can be parsed to generate a rucksack assessment of the player’s training. In a game sense this is his “score”, and being able to assess training performance is very important to the USMC.

Having completed one cycle of flashcard interaction in an example in Section 4.2, the player may now review an assessment of the interaction by exiting the vignette, selecting the assessment tab in the “Rucksack” and clicking “Run Assessments” (Fig. 8). The player will see “Partially Trained” for CAOCL TCAT 005: “Professionalism (demonstrate professional behavior when interacting with locals, mindful of status, perspective, and perceptions of the people with whom they interact and are not condescending or demeaning; demonstrate respect for women by exercising deference as much as the situation allows: seek permission to enter homes when it is practical to do so).”

5 Conclusions and Next Steps

This article has described a virtual village we are currently assembling for the US Marine Corp. Called NonKin Village, this is a gameworld that brings life to agents of all factions in sort of a SimCity style of play although it supports street level interaction and dialog with agents to learn their issues, needs, grievances, and alignments and to try to assist them in countering the agenda of the insurgent faction.

On the technology side, this research thus far concerned how to merge three successful tools from our lab – a cognitive sim emulator (PMFserv), a social layer (FactionSim), and a story engine (AESOP). The goal was to develop a prototype synthesis that preserves the strengths of each tool. This synthesis was summarized in Section 2 in terms of the separate social and cognitive model suites that were integrated. Section 3 then explained the “baking process” that seamlessly integrates all data needs of both suites to a common village file. The theories contained in the default version of NonKin straddle individual psycho-physiological ones (stress, emotion, sacred values, etc.) as well as social ones (belonging, mobilization and grievance, group membership, motivational congruence). One can edit these starting theories and their parameter sets with the internal editors of FactionSim and PMFserv and with the AESOP editor module. As such, NonKin serves as a successful proof-of-existence test for a socio-cognitive agent architecture.

As Section 4 described, this exercise was successful in producing a first training synthesis, a proof of concept that a fuller featured synthesis could be accomplished. Today, one can play NonKin Village flashcards, LANES, and FEX levels and not realize that previously non-communicative technologies are now interoperable in the background.

To support NonKin, we successfully implemented first versions of several features that are now open for further research as we spiral toward the next version. In fact, there are a host of further research directions that a project like NonKin opens up. We mention but a few of these here.

An important goal of our research is to instantiate sufficient content in the models, objects, and markups of the NonKin generator so that it will be largely a matter of turning things off when trying to set up NonKin for a new village or region of the world. Our approach for doing that relies on filling the sim with social science theories about descriptive and profiling methods. Thus the PMFserv holds hundreds of models whose parameters have been tested and shown to pertain in applications for North and South Asia, Africa, the Mid-East, and so on. In addition, we are evolving object taxonomies for the range of objects mentioned in this paper – transgressions, events, interactions, etc. – that will support migrating the village to new settings. There is much to be done, but ultimately, we will add editors that permit non-programmers to readily turn off features not needed and to adjust others to the setting of interest. We have achieved this style of editing before with our Athena's Prism world diplomacy game and believe it is repeatable here.

In the expectation that NonKin will ultimately be fielded in remote locations, we have teamed with a company (IMC Corp) that is capable of supporting NonKin applications and users around the world. Also, they have begun looking into ways to automatically instantiate a village game directly from intelligence and event databases. If successful, this would turn NonKin into a recreation of actual rather than mock villages. That would make it useful for tactical purposes, a topic perhaps best reserved for a different paper.

To support the NonKin trainee, we also added a rucksack where the player keeps a journal and can do analysis on what has transpired (Section 4.3). This is also where the player can find copies of his current orders, dialogs-to-date held with all characters, after action reports, and feedback on his training thus far relative to pedagogical objectives for the village. A Colonel agent gives orders and a feedback agent exists – both of whom the player can converse with. Each of these player support elements are items of continual refinement.

As a final note, we have just begun a collaboration with a 3-D videogame authoring group who has an immersive world where the player interacts with individual agents to learn task- relevant foreign languages, gestures and basic matters of courtesy -- Alelo TLT‘s Tactical Language Training System [4]. The NonKin project hopes to benefit from the use of their services to provide its village interaction, cultural training, and COIN operations in an immersive 3-D rehearsal and training environment. This should enhance the immersive nature of the training experiences and help players to more rapidly transfer lessons to the real world.

Acknowledgement. The research on NonKin Village was supported by ONR/HPTE and USMC/TECOM. Some of the FactionSim related work was supported by AFOSR. The transition is being supported by the DOD/HSCB program. None of these agencies are responsible for the views expressed here.

References

1. Anon: The U.S. Army/Marine Corps Counterinsurgency Field Manual, FM3-24 (2007)
2. Chiarelli, P.W., Michaelis, P.R.: Requirement for Full-Spectrum Operations. Military Review (July-August 2005)
3. Griffith, S.B.: Mao Tse-Tung On Guerrilla Warfare. Praeger, New York (1961) (Translation)
4. Johnson, W.L.: Serious use of a serious game for language learning. In: Luckin, R., et al. (eds.) Artificial Intelligence in Education, pp. 67–74. IOS Press, Amsterdam (2007)
5. Knight, K., et al.: Transgression and Atonement. In: Dignum, V. (ed.) AAAI Workshop Proc., Chicago (2008)
6. Kilcullen, D.: Twenty-Eight Articles: Fundamentals of Company-Level Counterinsurgency. ISPERE – Joint Information Operations Center (2004)
7. Nagl, J.: Learning to Eat Soup with a Knife. University Chicago Press, Chicago (2002)
8. Petraeus, D.H.: Observations from Soldiering in Iraq. Military Review (2006)
9. Silverman, B.G., Johns, M., Weaver, R., Mosley, J.: Authoring Edutainment Stories for Online Players (AESOP). In: Balet, O., Subsol, G., Torguet, P. (eds.) ICVS 2003. LNCS, vol. 2897, pp. 65–73. Springer, Heidelberg (2003)
10. Silverman, B.G., Bharathy, G., O'Brien, K.: Human Behavior Models for Agents in Simulators and Games: Part II – Gamebot Engineering with PMFserv. Presence 15(2), 163–185 (2006b)
11. Silverman, B.G., Bharathy, G.K., Nye, B.: Eidelson: Modeling Factions for 'Effects Based Operations': Part I – Leader and Follower Behaviors. Journal Computational &Mathematical Organization Theory (2007a)
12. Sun, R.: Cognition and Multi-Agent Action. Cambridge Univ. Press, Cambridge (2004)
13. USMC, Small Unit Leaders Guide to Counter Insurgency Operations (COIN), USMC, Quantico (2006)
14. Zacharias, G.L., MacMillan, J., Van Hemel, S.B. (eds.): Behavior Modeling and Simulation: From Individuals to Societies. National Academies Press, Washington (2008)
15. Operational Culture Study Program- Operational Culture Guidance (DVD-ROM), USMC Center for Advanced Operational Culture, Quantico VA (2006)

On Evaluating Agents for Serious Games

Emma Norling

Centre for Policy Modelling
Manchester Metropolitan University
norling@acm.org

Abstract. With the recent upsurge in interest in agents for 'serious games,' there has been a focus on the 'believability' of agents in these settings. In this paper, I argue that when evaluating agents in this context, believability is often in fact of relatively minor importance, and indeed that focusing on this criteria can detract from the ultimate goals of the games. I present this argument in the context of a project for which the aim was to extend the BDI agent framework to better support human modelling in serious games.

1 Introduction

In games that are developed purely for entertainment, one of the key goals is to create 'believable' characters; this "does not mean an honest or reliable character, but one that provides an illusion of life, thus permitting the audience's suspension of disbelief" [1]. In other words, so long as the behaviour of the character isn't such that the viewer stops to question it, it doesn't matter if in reality the behaviour is implausible; what is important is that the viewer is carried along with the story, and that the character behaves 'sensibly' within the context of the story.

However 'serious games' is a label that has been termed to apply to a diverse range of gaming applications whose primary purpose is *not* entertainment. Sawyer and Smith have presented a taxonomy of these games, where the purpose ranges from from training and education through to advertising [2]. With these different goals, it is important to acknowledge that believability (as defined above) will not necessarily be the primary evaluation criterion for the characters within these games. For example in a training scenario, characters that will respond realistically – in terms of both physical and cognitive constraints – to a leader's instructions are more important than ones that simply engage the user in the story. If unrealistic behaviours in the characters go unnoticed by the trainee due to their believability, this may in fact ill prepare the trainee for the behaviour of real people in the real world.

The issues involved in evaluating agents that are designed to model human behaviour in serious games are explored in this paper using examples from a project that aimed to extend a BDI-based agent framework to better support human modelling [3]. It is important to note though that the focus here is on the evaluation of particular extensions to the framework. No knowledge of the underlying framework, or the details of the implementations of the extensions, is necessary in the discussion of how these extensions are evaluated. The motivation for the project came from the use of BDI-based models of human behaviour in military simulation applications, but the discussion of how agent-based models of human behaviour should be evaluated applies more broadly to other classes of serious games (and indeed, even to games that are purely for entertainment purposes).

The aim of this paper is then to clarify then how we should go about evaluating agents that are designed to model human behaviour in these serious games. Firstly, we must determine on what criteria we should evaluate these agents, as discussed in Section 2. The next step is to determine what type of evaluation is suitable for those criteria, as in Section 3. It is also important to recognise that the target system will not necessarily be the ideal testbed for this evaluation. The simulated worlds of serious games are often quite complex, and as discussed in Section 4 it may in fact be easier to evaluate agent behaviour in a more controlled specialised testbed. The two examples mentioned above are then introduced in section 5, along with the details of their evaluation, and the paper concludes with a summary of the key points raised.

2 On What Criteria *Should* These Agents Be Evaluated?

This question must be answered by considering two things: first, what is the aim of the application (for example training for a real-world scenario, or recruitment to a particular profession), and second, what is the purpose of the agents within this application (for example background 'colour' versus close team-mates). In some sense, the goal is to produce human-like behaviour, but what is meant by 'human-like' is going to vary depending on the answers to these two questions. Broadly speaking, it means *either* 'more believable' or 'more realistic,' where 'realism' here focuses on accurately capturing particular aspects of human behaviour, while 'believability' focuses on creating models that are superficially plausible, and do not detract from the overall flow of the game, but may have unrealistic behaviours and/or abilities.

It is important to note however that regardless of whether the focus is believability or realism, evaluations must sometimes rely on qualitative, and often subjective, judgements. Although some aspects of human behaviour can be precisely quantified – such as the time to react to a stimulus – others are more nebulous. For example when evaluating whether an agent's decision-making strategy (that is, choice of which set of actions to follow) is realistic, even though the agent might sometimes make an obviously *wrong* choice (in terms of obviously being unable to achieve its goal), this is not necessarily *wrong* in terms of behaviour, because people will also sometimes make the wrong choice. The question is really whether the agent's choices are *plausible*, which is a subjective judgement that must be made.

3 Performing the Evaluation

Cohen's book, *Empirical Methods for Artificial Intelligence* [4] is an excellent reference book when approaching the evaluation of agent-based models of human behaviour. It looks at the various types of statistical analysis that can be used to evaluate AI systems, explaining the assumptions of each and the conditions under which it is appropriate to use them. However the examples and discussions in this book assume that there is a quantifiable measure to work with. In some examples, such as the MYCIN case study [4, Sect. 3.1.2], the quantifiable measure is based on qualitative judgements, but as this qualitative judgement is reduced to a simple accept/reject decision, discarding most of the information. He then goes on to perform statistical analysis of this result.

However a common mistake – not one made in Cohen's book I hasten to add – when translating qualitative data into numerical results is to assume an underlying linear relationship, which rarely exists. The most common occurrence of this mistake is with the use of Likert scales. Likert scales are typically used thus: questionnaire respondents are presented with a series of statements, and are asked to give a response on a five point scale, with for example '1' on this scale being 'strongly disagree', '3' being 'neutral' and '5' being 'strongly agree.' The number of points on the scale can vary, although there is usually an odd number (allowing for a neutral response). The questions may also vary, so that, for example, the numbers might represent the scale 'much worse' to 'much better,' or indeed anything that could be expressed on such a comparative scale. The problem arises when statistical analysis *that assumes a linear relationship between the points on the scale* is applied to the results. Although it is safe to assume that respondents will treat it as an ordered scale, it should not be assumed that, for example, the difference between "strongly disagree" and "disagree" is the same as the difference between "disagree" and "neutral." Nor is it safe to assume that one person's choice of "agree" is equivalent to another's. Indeed rankings may actually be lexicographic (as in Cohen [5]) rather than numeric. So for example if respondents are asked to rate features of an item, rated from "very unimportant" to "very important," an object with three "very important" features might be more desirable than *any* object with only two "very important" features, regardless of the relative numbers of "important" features.

What is the significance of this? Unless this condition of strict linearity holds, few statistical manipulations of the numerical values are valid. If it does not hold, it is for example meaningless to report an 'average score for respondents'. The respondents can be instructed that they should treat the scale as a linear one, but it is important to recognise that simply by adding labels to points on the scale, their meaning is loaded. It may be more appropriate to simply ask the respondents to 'rank the importance of each feature, on a scale of one to five.' The alternative is to collapse the scale into a binary choice (as in the MYCIN analysis), usually around the midpoint of the scale. In this case, standard techniques can be applied.

4 Requirements of a Testbed

When developing agent-based models of human behaviour in games, each model is often a relatively small component in the overall application. The game engine itself usually provides a rich simulated world, and there are often many other agents with which to interact. This can make it difficult to conduct controlled, repeatable experiments within the application, meaning that it may be appropriate to use an alternate testbed for evaluating the models.

Whatever measure is being used for evaluation, be it qualitative or quantitative, it is also important to consider how the models will be constructed and how they will be judged. If domain knowledge will be required to construct the models, the experts who will supply this knowledge should be readily accessible. If the model results are going to be directly compared against human data, are there existing data sets available, or will this data need to be collected for comparison? In the case of qualitative assessment, who is qualified to make that assessment, and are these people readily available?

The three essential characteristics of an environment for testing and evaluation are then

1. That it provide the means and opportunity for the model to perform appropriate tasks to demonstrate the desired features,
2. That there be easy access to the experts who perform these tasks that will be modelled, to facilitate development of the models, and
3. That the impact of the desired features can be measured in some way, and if qualitative judgements must be made, that suitably qualified judges are available.

This last point has only been touched upon so far, but is important. It may be possible to draw from the same set of experts as would be drawn upon when gathering knowledge to construct the models, but in some cases these may not be the best judges of whether or not behaviour is more human-like. An alternative is to use behaviour experts – people who are expert at analysing human behaviour, but not necessarily expert at the task being performed. In this second case, it is important that the task is not too highly specialised, so that the behaviour experts can understand and judge it.

As well as these essential characteristics of the testing environment, there are a number of additional characteristics to consider. The characteristics in this second set are *not* essential characteristics; each must be considered in the context of the features to be demonstrated. If the characteristic is necessary for the demonstration, it should be included, and if it is not, it may be desirable to avoid it, since it may confound results. These characteristics are:

– That there be multiple means of achieving tasks, giving the agent the opportunity to choose,
– That the environment be only partially controllable, so that the outcome of an agent's action is not guaranteed,

– That the environment be rich, with many sensory inputs (not necessarily relevant to the task), and many different types of action that can be performed,
– That the world contains stressors and moderators (physical or mental) that can act upon the agent,
– That the agent is forced to make short term time-critical decisions as well as long term strategic decisions,
– That the environment supports extended time frames, and
– That there are other autonomous entities with whom the agent may interact.

It is possible – indeed likely – that after identifying the desired characteristics, there may be one or more existing environments that could be used as testbeds (see Section 4.1), and of course there is always the option of developing a tailored environment as a testbed (Section 4.2).

4.1 Avoiding the Pitfalls of Pre-existing Environments

Pre-existing environments are used in two ways. They may be the core part of an existing application, for which new models of human behaviour are being developed. Alternatively, as mentioned above, an existing environment may be identified as a suitable testbed for a new or modified framework. Whether the existing environment is being used because it is part of the target application or because it is deemed a suitable testbed for a framework, a number of issues can arise due to the lack of control over the agent-environment interface.

Perception. Problems with perception are not limited to interfaces having graphical representations. Even when features of the environment are labelled, they are not necessarily labelled appropriately from the agent's perspective. For example, if a tree is simply labelled with its position, in the form $(tree, (x, y, z))$, there are many unknowns. Is it big enough to hide behind? Can it be climbed? The way in which it has been labelled means that the agent has no way of telling, although if a human accessed the environment through a graphical interface, he/she would be able to use the visual information to answer these questions. Of course these questions are only important if the subject being modelled makes reference to these features – if the only thing that the subject does with trees is go around them, then a simple label with the position is sufficient.

Another problem is that the environment doesn't always provide the *right* sense data at a given point in time. In some cases the environment does a calculation of what the agent should be able to see and sends only that information, but in other cases that is left to the agent. The most common situation is a compromise. The environment sends a limited set of information, but that set is still more than the agent should really be able to sense – a first pass estimate, if you like. The agent must then make a second pass to determine which of this data it can *really* see or hear.

If such problems are encountered, what can be done? In the case of under-representation – for example if the agent wants to know if a tree is big enough to hide behind but it is only labelled with position – the only fix can be on the environment side. If there is no means of changing the environment, and hiding behind trees is an essential element in the agent's performance of the task, the chosen environment is inadequate. However if the problem is that the information is available but difficult to interpret it may be possible to handle on either side of the interface. If it is not possible to adjust the environment, it should be possible, given appropriate resources, to perform the necessary processing in the agent. It comes down to an engineering decision: is it easier to label the environment, or to let the agents 'see'? There is no easy answer to that question, in part because the ease of labelling the environment is going to depend upon the environment being used. (See the Heinze's doctoral thesis for a detailed exploration of this issue [7].)

Action. There are related problems that can arise with the actions to be performed in the environment, largely due to the coarse granularity of actions that can be performed. When the agent is interacting with the same interface as the human, sending simulations of the same mouse movements or key clicks that a human player would (as in both of the testbeds presented in Section 5), this issue is avoided. This are relatively unusual cases, where the models were actually performing the same tasks as the human. (That is, the agents were operating in the *exact* environment as the subjects that they were modelling.) More commonly, the models are performing a *simulation* of the tasks that the humans perform, and the actions that they are able to perform are modelled at a far coarser granularity than mouse movements and key clicks.

In an environment involving movement over terrain for example, there are a variety of ways in which the interface could accept movement specification, such as:

1. Move to a specified point.
2. Move in a specified direction for a specified amount of time.
3. Move a single step, or turn.

Each of these three alternatives gives a different amount of control over the path to the agent. In the first case, the path planning is handled entirely in the environment, and there may well be no way for the agent to interrupt the movement. In the last case, the agent has complete control over its path, but must send *many* more commands via the interface. It may be possible to decompose 1) or 2) into the third case by issuing commands for a series of very short movements, but the granularity of the environment itself sometimes restricts this. It may only be possible to move to a series of fixed points in the environment, making single steps impossible in some cases.

Like the under-representation problem in perception, this problem is one that can only be dealt with on the environment side of the interface. If there is no chance to modify the environment, it is essential to determine the granularity of actions that will be performed early in development. If it is essential to be able to perform actions at a finer grain than is supported by the interface, an alternate environment must be found.

Another problem that can occur is that entities may not be able to *use* specific objects or features in the environment in the that they would expect. For example, there may be ledges that *should* allow handholds but do not, or small rocks that can't be picked up. Such a problem will be familiar to any computer game player, who was frustrated at not being able to climb a wall, or not being able to use a flaming torch as a defensive weapon. In the world of military simulation the problem is less common (perhaps because aesthetics are less of a concern, so there is less 'clutter'), but does crop up on occasion. In this case, it is possible to argue that the problem is actually a bug in the environment model, and there is a chance that the developers will fix it. If not, once again an alternate environment must be found if the action is essential to the task.

Timing. The timing of pre-existing environments can also sometimes cause problems. Even when environments operate continuous time (as opposed to stepped-time), entities that connect to the environment usually receive sense data and have the opportunity to perform actions in fixed time slices. For most purposes, these time slices are frequent enough as to be effectively continuous (in the order of a few hundred milliseconds), but for applications where timing is critical, it is important to compare the timing of the environment with that required by the agent.

4.2 (The Problem with) Building Tailored Environments

With all these potential problems with pre-existing environments, it might seem that the solution would be to create tailored environments whenever possible. In such environments, the agent-environment interface can be carefully designed so as to avoid the aforementioned problems. The difficulty in this approach is the magnitude of the task. The pre-existing environments that have been mentioned – be they game engines or military simulation environments – have been developed by teams of programmers over several years. A tailored environment may take somewhat less than this, as unnecessary detail can be omitted, but there is still a huge amount of work required to create an environment of the complexity that would be required.

A sample tailored environment was developed for the very early stages of developing the decision-making strategy extension to the BDI framework that is mentioned in Section 5. The aim of that work was to provide an extension that gave the framework a more realistic decision-making strategy. In order to demonstrate this extension, the environment needed to provide sufficient detail that models built using the extended framework could exercise the enhanced decision-making strategy. In order to do this, the following scenario was considered:

> Suppose an agent has a goal of getting from one side of the business district of town to the other (a distance of about 3km). She may have three plans for handling this goal: the first involves walking; the second involves driving; and the third involves catching the train. Each of these

> plans has a context. The walking plan might specify that it is only applicable if the agent has at least 40 minutes to achieve the goal, and that it is not applicable if it is raining heavily. The driving plan might indicate it is only applicable if the car has fuel. And the context of the train plan might be that it is only applicable between 6am and midnight (trains don't run all night).[8]

The world required to support this scenario is extremely limited, and it it provided only the barest minimum for the preliminary testing of the decision-making strategy. Nevertheless, the effort required to construct this simulated environment was considerably more than that required to construct the models that inhabited it. The experience gained from developing this simple environment largely contributed to the decision to use a pre-existing environment for future work.

4.3 Back to the Target Application?

Given the above-mentioned problems with both alternative existing environments *and* purpose-built environments for testing, does it not make more sense just to stick with the target application? The answer to this is a qualified 'yes.' Because of the difficulty in controlling the experiments in many games, it often makes sense to perform initial experiments and sensitivity analysis in a more simple and constrained testbed. However particularly when it comes to the more subjective judgements that are sometimes required, which can only be made in similar scenarios to those that arise within the target application, it is generally this application that should be used. The pitfalls above might be encountered when developing the models for the application, and some sort of workaround must be incorporated into the model (or the application itself modified). However in these cases the pitfalls *must* be addressed for the final system in any case, whereas if they are encountered in an alternative environment, considerable effort might be needed to address them when they are not a problem in the target system.

5 Two Cases in Point

These issues were highlighted in the previously-mentioned project, that aimed to extend a BDI-based agent framework to better support human behaviour modelling [3]. The work was motivated by the ongoing use of BDI-based models of human behaviour at Defence Science and Technology Organisation (DSTO), Australia (for example [9], [10], [11], [12], [13] and [14]). Rather than developing agent-based models of human behaviour for a particular application, the aim was to demonstrate how the BDI framework could be extended to better support human behaviour modelling. To illustrate the methodology that was proposed, two particular extensions were implemented, one providing a limited model of perception and action, and another providing an alternative decision-making strategy to agents.

Because these extensions were not developed with a particular application in mind, there was no target application available in which they could be evaluated, and so it was necessary to choose suitable testbeds for evaluation. Each extension was evaluated in isolation (and in fact technical issues prevented the full evaluation of the decision-making strategy extension), and the analysis of the needs for evaluation of each extension meant that very different testbeds were used.

5.1 Evaluating the Perception and Action Extension

The perception and action extension to the framework that was implemented included only limited capabilities, allowing the model to interact with a mouse-driven, java-based interface, assuming that the 'user' is seated directly in front of the computer screen in an optimal position for viewing. In order to test and evaluate this extension, a simple telephone-dialling task was developed, along the lines of one used by Ritter et al when working on a similar extension for Soar [15].

Fig. 1 shows two different telephone interfaces with which the models and human subjects would have to interact. The different button sizes in the two interfaces should lead to different times for telephone dialling tasks, because the smaller the buttons, the more accurate the movements must be. The subject (human or model) was required to read instructions from the top line of the interface, each of which instructed him/her/it to dial a particular number, then click OK.

The telephone numbers that the users were instructed to dial were numbers that they knew well, such as their home or office numbers. The idea was that the users should not have to think about the numbers – doing so would add time to the task of the human subjects that was not accounted for in the agent models. Because this meant that different subjects dialled different numbers, and some numbers are naturally more easy to dial than others (for example 5-5-5

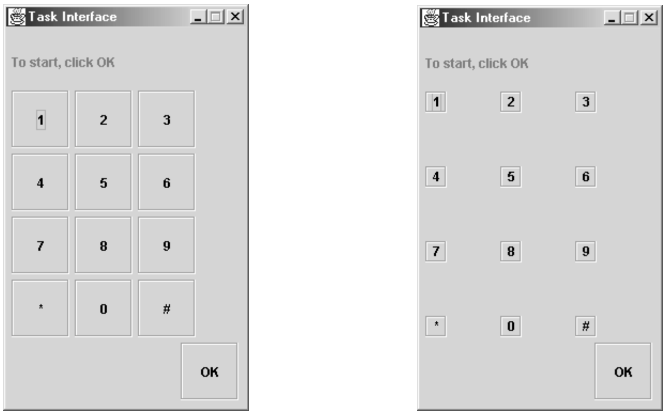


Fig. 1. Two sample interfaces with which the model and human can interact

versus 1-9-4), models were compared against individuals on a case-by-case basis, although the only difference in each model was the numbers that it used for dialling.

Although this was a very simple task, the invariance of perception and action over tasks meant that if there was a correlation between human and model results on this task, the results should also hold for more complex tasks.

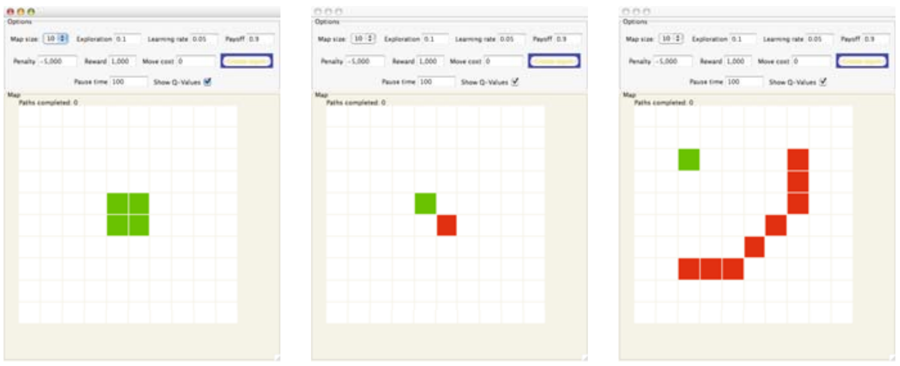
5.2 Evaluating the Decision-Making Extension

As has already been mentioned in passing, evaluating the decision-making extension was somewhat more complicated than evaluating the perception and action extension. One point is that the extension provides one new form of decision making to the agent, but it is freely acknowledged that in real life, people use a range of decision-making strategies, with recognition-primed decision making (the strategy implemented in the extension) being just one of them. Thus one wouldn't expect an agent implemented with this extension to make strategies *exactly* as a real person would. It should however produce a model that makes more human-like decisions than one built using the standard framework. There could be some decisions that are still obviously 'wrong,' in that a real person wouldn't make them, and there will probably be some decisions that are 'wrong' in that they are sub-optimal (just as a real person makes sub-optimal decisions).

Preliminary work on this extension used a simple testbed that required the model to choose between different strategies for getting across town. The experience of developing this testbed, coupled with the realisation that a far more complex environment would ultimately be needed to evaluate the extension lead to the decision to find a suitable pre-existing environment for this evaluation. Military simulation environments were the target area of application, but access to these is obviously restricted, so an alternative was sought. The solution was to use a commercial game engine (in this case, Quake 2) for the testbed. However although this game engine provided a rich environment for observing behaviour, it was not possible to carefully control situations and events. Thus a more simple environment was also used for the sensitivity analysis of the implementation (which included a number of free parameters). This simple environment did not allow the agents to display the full richness of human behaviour, but did allow controlled experiments to examine the parameter settings.

Sensitivity Experiments. The experimental testbed for this stage was a simple path-finding task in a gridworld, as shown in Figure 2. The aim is for the agent to find a path from any point on the grid to a goal (green) square, while avoiding the anti-goal (red) squares. The configuration shown here is a sample one – the grid can have any combination of penalty or reward squares. Moreover, it is possible to change this configuration at run-time, and observe how the agent behaves in a dynamic environment. The agent's goal is to find a shortest path to the goal square, from any point on the map. The task itself is modelled upon one used by Masson to demonstrate Q-learning [16].

As Masson explains in the description of his Q-learning demo, "it would be silly to build a path finder using Q-learning" [16]; for this problem there are



Map A Map B Map C

Fig. 2. Three sample maps in the gridworld testbed

specifically-tailored techniques that are more appropriate. However this somewhat artificial demonstration does allow exploration of the parameters, and has some correspondence to the real world decision problems that this extension is designed to deal with. Firstly, the number of options at any point is about right: faced with a particular goal, a person typically does not consider a wide range of possible options, usually no more than three [17]. Secondly, the idea of there being certain 'good' states, certain 'bad' states, but a large number of indeterminate states is also true of real-world problems. Moving one step closer to a goal square is equivalent to completing a sub-plan in a real-world problem: it might get you that little bit closer to the ultimate goal, but unless it has unexpected consequences (good or bad), it is usually fairly insignificant in the scheme of things.

The Testbed for Qualitative Evaluation. In order for the 'standard' and 'enhanced' models to be compared, the testbed needed to exercise the enhanced features, that is the modified decision-making strategy, in the enhanced framework. The decision-making strategy that was implemented (recognition-primed decision making [18]) is one that is typically used by experts operating in particular types of environments, namely, ones with the following characteristics:

– Ill-structured problems
– Uncertain, dynamic environments
– Shifting, ill-defined or competing goals
– Action-feedback loops
– Time stress
– High stakes
– Multiple players

In other types of environments, or for people operating outside their area of expertise, other decision-making strategies are likely to be used, and so it was important that the testbed demonstrate these characteristics. Many of the environments used in military simulation *do* display these characteristics, but due to the restricted access to both the environment and the experts to be modelled these were inappropriate environments for this work.

Action games were identified as a suitable alternative to these military simulation environments. They had the benefit of being readily available, with many local and easily accessible experts, while at the same time having the above characteristics (with some variation from one game to another). There were several possible game engines that could have been used; Quake 2 was was selected because it had already been used for similar research (for example in Laird's work [19]), considerable information regarding its working were available on the web (facilitating agent connections to the engine). At the time this project commenced, the most significant alternative, the GameBots engine [20], was not yet public, and in any case the two environments are very similar.

Using a commercial game engine has several advantages. First of these is that the worlds of these engines are rich, dynamic worlds, which the researchers can use rather than creating their own. Moreover they are worlds in which people (players) regularly operate, so there are many subjects who can be modelled. Even people who have never played the games usually have some concept of what is involved in the game, and so the models themselves are meaningful to a wide range of people. And the numerous players are available as critical judges of the models, as well as subjects to be modelled.

The Proposed Qualitative Evaluation. Technical difficulties that arose when interfacing the agent with the Quake 2 engine prevented this proposed evaluation from being completed, but it is presented here to illustrate how a qualitative evaluation could proceed. The plan was to have experienced Quake 2 players evaluate the models. These players would be invited to play against models built using both the standard (unmodified) and enhanced frameworks. They would give their feedback via a written questionnaire that included both open-ended and Likert-scaled responses.

In the proposed evaluation, the evaluators would be split into two groups. The first group would play first against the models built with the standard framework and then against models built with the enhanced framework. The second group would be a control group, playing against models built with the standard framework in both cases – although they would still be told that the models in the second game were built using the enhanced framework. The intention here would be to detect whether the belief that they were using an enhanced framework would influence the way that the players judged the models.

A further variation that could provide another control case would be to have a third group of evaluators who played first against models built with the standard framework and then against the actual players who were modelled. Rather being told that they were doing this, they would be told that they too were playing against models built using an enhanced framework, and presented with the same questionnaire as in the first two groups. The aim with this group would be to detect how well the players were able to detect changes in play. The downside of such a case would be the commitment needed by the players who served as subjects – they had already been involved in several hours of interviews during the knowledge elicitation process, and this evaluation would require several hours involvement in game play (each game in the evaluation taking twenty minutes). To avoid overburdening these volunteers, an alternative would be to run a sequence of games involving standard models, enhanced models and the human subjects, and recording these games. The evaluation would then involve viewing replays of these games, rather than playing against the models and subjects. It is not clear however whether the game players would be able to make the same judgements from watching the replays as they would from actually participating in the games.

The questionnaire itself (which is included as an appendix in [3]) consists of three parts: one to gather information about the evaluator, one about their judgement of the 'standard' models, and the final about their judgement of the 'enhanced' models. The questions were a mixture of free-form responses and responses constrained by Likert scales. The free-form responses were intended to provide a basis for future work (such as what characteristic should be considered for the next enhancement) as well as detailed feedback about the implemented enhancement.

Two questions with Likert scale responses are included in the questionnaire. The first asks the respondent to compare the standard models with their conception of an expert human player, and the second asks them to compare the 'human-ness' of the models in the first game with those in the second. Given the discussion on the pitfalls of the use of Likert scales in Section 3, what justification is there for the inclusion of these questions? The answer is that as in Cohen's description of the MYCIN evaluation, the responses would be reduced to binary decisions for the purposes of statistical manipulation. So the responses to the first of these questions would be placed into one of two categories: 'equal to or better than an expert human player' and 'worse than an expert human player.' For the second of these questions, the categories would be 'less or equally human-like' or 'more human-like.' These binary distributions could then be used for analysis and comparison between the two groups of respondents.

Why then ask respondents for the finer-grained distinction that is provided by a Likert scale? The answer is that there is more information in these responses than in a binary response, although admittedly there may be more information still in the open-ended responses. The full detail from a Likert scale responses may not be used for the statistical analysis of the feedback, but it can be useful in the qualitative analysis, particularly as a precursor to a related open-ended question (as in the case here), where it can focus the responses. These responses to these questions might also be used to formulate more directed questions for focus groups as part of ongoing research.

6 Conclusions

The purpose of this paper has been to highlight the fact that when considering models of human behaviour in serious games, believability is not necessarily a primary evaluation criterion – and in some cases believability can even detract from the primary goal of the application (for example giving 'false training'). Thus before evaluating any agent-based models of human behaviour in these types of applications, it is first necessary to consider what is the aim of the application concerned, and then what is the role of the models of human behaviour within them. In many cases, it will be more important for models to behave *realistically* in particular ways than to produce behaviour that is overall believable.

When considering realism of particular aspects of behaviour, the approach to evaluation will depend upon the aspect(s) of interest. Some can be compared with human performance in a quantifiable way, which in general leads to relatively straightforward evaluation. However other aspects of human performance, such as the decision-making strategies mentioned here are not easily quantifiable, and in such cases, qualitative evaluations are required. The two examples presented in Section 5 illustrate how one may proceed in each of these cases.

References

1. Bates, J., Loyall, A.B., Reilly, W.S.: An architecture for action, emotion, and social behavior. In: Castelfranchi, C., Werner, E. (eds.) MAAMAW 1992. LNCS, vol. 830. Springer, Heidelberg (1994)
2. Sawyer, B., Smith, P.: Serious games taxonomy. Presented at the Game Developers Conference, San Francisco, CA, USA (February 19, 2008)
3. Norling, E.: Modelling Human Behaviour with BDI Agents. PhD thesis, The University of Melbourne (forthcoming, 2009)
4. Cohen, P.R.: Empirical Methods for Artificial Intelligence. MIT Press, Cambridge (1995)
5. Cohen, P.R.: Heuristic Reasoning about Uncertainty: An Artificial Intelligence Approach. Pitman, London (1985)
6. Epic Games: Unreal Technology - Unreal Engine 2 (2000–2008), `http://www.unrealtechnology.com/features.php?ref=past-versions`
7. Heinze, C.: Modelling Intention Recognition for Intelligent Agent Systems. PhD, The University of Melbourne (2003)
8. Norling, E., Sonenberg, L., Rönnquist, R.: Enhancing multi-agent based simulation with human-like decision-making strategies. In: Moss, S., Davidsson, P. (eds.) MABS 2000. LNCS (LNAI), vol. 1979, pp. 214–228. Springer, Heidelberg (2001)
9. Murray, G., Steuart, S., Appla, D., McIlroy, D., Heinze, C., Cross, M., Chandran, A., Raszka, R., Tidhar, G., Rao, A., Pegler, A., Morley, D., Busetta, P.: The challenge of whole air mission modelling. In: Proceedings of the Australian Joint Conference on Artificial Intelligence, Melbourne, Australia (1995)
10. McIlroy, D., Heinze, C.: Air combat tactics implementation in the Smart Whole AiR Mission Model (SWARMM). In: Proceedings of the First International SimTecT Conference, Melbourne, Australia (1996)
11. Tidhar, G., Heinze, C., Selvestrel, M.: Flying together: Modelling air mission teams. Applied Intelligence 8(3), 195–218 (1998)
12. Heinze, C., Goss, S., Josefsson, T., Bennett, K., Waugh, S., Lloyd, I., Murray, G., Oldfield, J.: Interchanging agents and humans in military simulation. AI Magazine 23(2), 37–47 (2002)

13. Watson, M., Lui, F.: Knowledge elicitation and decision-modelling for command agents. In: Palade, V., Howlett, R.J., Jain, L. (eds.) KES 2003. LNCS, vol. 2774, pp. 704–713. Springer, Heidelberg (2003)
14. Goss, S., Heinze, C., Murray, G., Appla, D.: Agent enabled simulation, modeling and experimentation. In: Proceedings of the Eight International SimTecT Conference (2004)
15. Ritter, F.E., Baxter, G.D., Jones, G., Young, R.M.: Supporting cognitive models as users. ACM Transactions on Computer-Human Interaction 7(2), 141–173 (2000)
16. Masson, T.: Artificial intelligence – the Q-learning algorithm (2003), http://thierry.masson.free.fr/IA/en/qlearning_about.htm
17. Ross, K.G., Klein, G.A., Thunholm, P., Schmitt, J.F., Baxter, H.C.: The recognition-primed decision model. Military Review LXXXIV(4), 6–10 (2004)
18. Klein, G.A.: A recognition-primed decision (RPD) model of rapid decision making. In: Klein, G.A., Orasanu, J., Calderwood, R., Zsambok, C.E. (eds.) Decision Making in Action: Models and Methods, pp. 138–147. Ablex Publishing Corporation, Greenwich (1993)
19. Laird, J.E.: It knows what you're going to do: Adding anticipation to a quakebot. Technical Report SS-00-02, AAAI (2000)
20. Gamebots: Gamebots homepage (2002), http://gamebots.sourceforge.net/

A PDDL-Based Planning Architecture to Support Arcade Game Playing

Olivier Bartheye and Éric Jacopin

MACCLIA, CREC Saint Cyr, Écoles de Coëtquidan, F-56381 GUER Cedex
{olivier.bartheye,eric.jacopin}@st-cyr.terre.defense.gouv.fr

Abstract. First, we explain the Iceblox game, which has its origin in the Pengo game. After carefully listing requirements on game playing, the contents of plans, their execution and planning problem generation, we design a set of benchmarks to select good playing candidates among currently available PDDL-based planners. We eventually selected two planners which are able to play the Iceblox video game well and mostly in real time. We describe both the predicates and some of the operators we designed. Then, we give details on our planning architecture and in particular discuss the importance of the generation of PDDL planning problems. We wish to report that no planner was tweaked during the benchmarks.

1 Introduction

On one side, the quality of the Artificial Intelligence is part of today's game evaluations and on the other planning systems still wait for going mainstream. Could a close encounter of both be fruitful? Over the years planning has occurred in the video-game domain with some success (e.g. [1]), with the work on the Goal Oriented Action Planning architecture being probably the most sucessful [2]. This time, we would like video-gaming to occur in the planning domain: designing and implementing a game and connect it to today's planners. First, thanks to the International Planning Competition [3], today's planners all accept planning problems written in PDDL [4] which is going to ease the selection of a good planner for video-gaming. Then, it is not only a matter of constructing a plan but also to generate PDDL planning problems from a video-game situation. Third, it is also a matter of executing plans in a video-game. And finally, what shall happen when the dynamics of video-games makes useless the plan currently executed? These matters and questions have rarely been addressed in the case of the close encounter of planning and video-games and it is the purpose of this paper to report the sparks.

This paper is organized as follows. We begin with a description of the Iceblox arcade game and argue why this game is a good choice for testing the performances of PDDL-based planners coping with planning problems from this domain. The next section presents and discusses the necessary requirements on a planning system playing an arcade game such as Iceblox. Then, we report on the selection process of available PDDL-based planners, from a wide perspective (simple Iceblox situations, several planners, various planning problem properties) to a narrow perspective where two planners were tested against more realistic Iceblox situations. Finally, we outline the main procedures of our arcade game playing architecture: the generation of PDDL planning problem and the plan execution process and discuss the importance of path planning.

2 Iceblox

Description. In the Iceblox video game [5], the player presses the arrow keys to move a penguin (named Pixel Pete) horizontally and vertically in rectangular mazes made of ice blocks and rocks, in order to collect coins. But flames patrol the maze (their speed is that of the penguin) and can kill the penguin in a collision; moreover, each coin is iced inside an ice block which must be cracked several times before the coin is ready for collection. The player can push (space bar) an ice block which will slide until it collides with another ice block, a rock or any of the four side of the game. A sliding ice block stops when it collides into another ice block, a rock or any of the four sides of the game and kills a flame when passing over it. If the player pushes an ice block which is next to another ice block, a rock or a side of the game, it cannot slide freely and shall begin to crack. Once cracked, an ice block cannot move any more and thus pushing a cracked ice block eventually results, after seven pushes, in its destruction. An ice block which contains a coin slides and cracks as does an ordinary ice block. A coin cannot slide; it can only be collected once revealed after the seventh push. The player gets to the next, randomly generated, level when all coins have been collected. Killed flames reappear from any side of the game and from time to time, as in Figure 1, it happens that locking the flames is a better strategy than killing them: since there is no constraint on time to

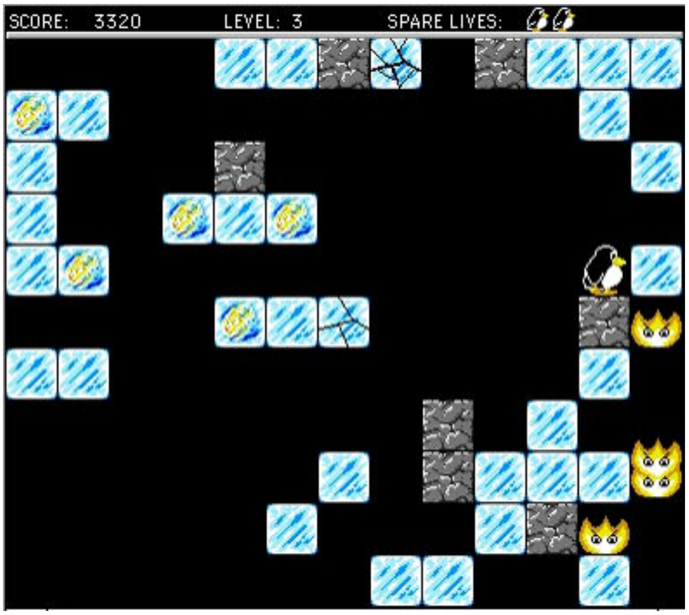


Fig. 1. An Iceblox game in progress

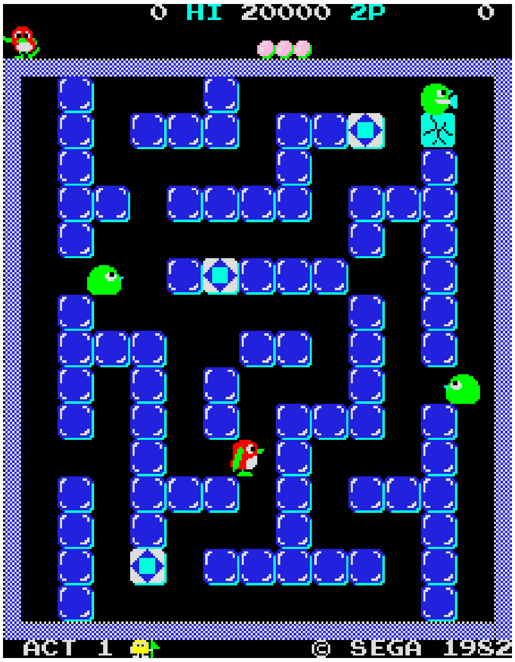


Fig. 2. A Pengo game in progress [7]

finish any level of Iceblox, this strategy gives the player all the latitude to collect coins. The number of flames and rocks increase during several levels and then cycle back to their initial values. Finally, the player gets 200 points for each collected coin, 50 points for each killed flames and begins the game with three spare lives.

Why Iceblox?. Iceblox is an open and widely available java implementation [6, pages 264–268] of the Pengo arcade game, published by Sega in 1982 (see Figure 2). In Pengo, there are pushable and crackable ice blocks but neither rocks nor coins; the main objective of the game is to kill all the bees patrolling the ice blocks mazes. Bees hatch from eggs contained in some ice blocks; the destruction of an ice block containing eggs results in the destruction of these eggs. In Pengo, bees can be killed with a sliding ice block and by a collision with the penguin when they are stunned. Pushing a side of the game when a bee is next to this side shall stun the bee for some short time. The player is given 60 seconds to kill all the bees of any of the 16 levels of the game (after the sixteenth level, the game cycles back to the first level). Bees accelerate when reaching the 60 seconds time limit and the player is given a short extra time to kill them before they vanish, thus making the level impossible to finish. There also are pushable diamonds which cannot be collected but must be aligned to score extra points. As in any arcade game of the time, there are many bonuses, making the high score a complex goal; and both the killing of the bees and the alignment of the diamonds score differently according to the situation. We refer the reader to [7] for further details about the game.

Iceblox obviously is easier than Pengo: no time constraint, uniform speed and scoring, no bonuses,... Main targets (coins) fixed rather than moving (bees or flames), flames cannot push ice blocks and fighting them is optional,... However, the locking of flames in a closed maze of ice blocks and rocks possesses a geometrical spirit similar to that of the alignment of diamonds in the Pengo game.

Consequently, we chose Iceblox because it is a simpler start in the world of arcade video games than the commercial games of the time.

3 Requirements

Game playing. On its way to collect coins, the planning system playing the Iceblox game shall badly either fight or avoid flames: disorganized movements or paths longer than necessary shall be allowed as long as they do not prevent Pixel Pete from collecting coins. Both the fighting and avoidance of flames shall be realized in (near) real time: game animation might look sometimes slow or sometimes irregular but shall never stop; in particular, flames shall always be patrolling the maze, even when traditional problem solving (that is, Planning) shall take place.

Plans (what's in a plan?). A plan shall be a set of partially ordered operators which represent actions in the Iceblox domain. What kind of actions shall we allow to be part of a plan?

Let us distinguish five kinds of actions that may happen in an arcade video game such as (Pengo or) Iceblox:

1. Drawing a frame to animate a sprite; this is the pixel kind of action. Animation always takes place, even if the player does nothing: for instance, a flame is always animated in the Iceblox game, even in the case where it cannot move because it is surrounded by ice blocks.
2. Basic moving of one step left, right, up or down and pushing an ice block. This is the sprite kind of action which is not interruptible for a certain number of pixels, usually the number of pixels of a side of the rectangle where the sprite is drawn (in the case of Iceblox, sprites are squares of 30 by 30 pixels). Each action of this kind exactly corresponds to a key pressed by the player: arrows keys to move left, right, up and down and the space bar to push an ice block.
3. Moving horizontally or vertically in a continuous manner, that is making several consecutive basic moves in the same direction; this is the path-oriented kind of action and means following a safe path in the maze.
4. Avoiding, fleeing, fighting or locking the flames in a closed maze of ice blocks and rocks. This is the problem solving kind of action and concerns survival and basic scoring. Ice block cracking until destruction to collect a coin also is a problem solving kind of action.
5. Defining goals and setting priorities on them; this is the game level strategy kind of action and is oriented towards finishing the game with the highest score. In the

case of Iceblox, this means ordering the collection of coins, noticing the opportunity to lock the flames in a maze and setting its importance in finishing the level; or else seizing the opportunity to seek cover behind ice blocks of a geometrical configuration created during game playing.

Out of the five kinds of action, only push actions of kind 2, abstract Moves based on rectilinear moves of kind 3 and the coin collection action of kind 4 shall be represented as operators.

The Plans just described are *not* as simple as they may appear: it is *not* the case that either their construction shall be too simple or their use shall be redundant with the player's actions. First, these Plans look like the plans for the storage domain which is part of the deterministic IPC benchmarks [3]. Second, as Table 1 shows, current planners agree with their non easiness. Third, there are Plans (both constructed and executed [8] or only executed [1]) in commercial video-games which are shorter and simpler.

Plan execution. The plan execution system receives a plan from the Planning system in order to execute it in the Iceblox game. Execution of a plan means executing the players actions (that is, actions of kind 2) corresponding to the operators of the plan, in an order compliant with the partial order of the plan. For instance, the plan execution system shall decide of several basic moves actions to execute a Move operator in the plan; which can include ice block pushing to facilitate the realization of the Move operator. The plan execution system shall also decide of taking advantage of the current Iceblox situation to avoid pushing an ice block when a flame is no longer aligned with it, to choose a new weapon or to avoid unexpected flames. These decision shall result from a local analysis and no global situation assessment shall be realized to take such advantages. The plan execution system shall eventually warn the user when it terminates (whatever the outcome, success or failure). The player can terminate the plan execution at any time by pressing any arrow key or else the space bar.

In case of emergency situations (e.g. no weapon is locally available, fleeing seems locally impossible, etc), the plan execution system shall call for re-Planning.

Planning. Planning refers to the plan formation activity. The player shall intentionally start the planning activity by pressing a designated key (e.g. the "p" key). Once running, planning shall not stop the Iceblox game: flames shall patrol the maze and sliding ice blocks shall continue to slide while the plan formation activity is running. The user shall be warned when the activity ends, either with success or else failure (e.g. a different sound for each case). If any of the arrow keys or else the space bar (push) is pressed before the end of the planning activity, then it is terminated. The user shall be able to play at any time.

Planners. The main objective of the work reported in this paper is to determine whether any of today's planners is able to play the Iceblox game: no planner shall be specifically designed to play Iceblox and no existing planner shall be tweaked to play Iceblox.

Due to the on-going effort of the International Planning Competition (IPC) [3], many planners are available today and most of them accept Planning data (i.e. states, operators) written in the PDDL language. Because of this wide acceptance, the overhead of both generation and processing of PDDL shall first be ignored; writing directly to a planner's data structures shall be considered only if game playing requirements are not met. Consequently, the Planning activity in the Iceblox domain shall take as input an initial state, a final state and a set of operators all written in the PDDL language.

Available planners are in general more ready to process PDDL text files than ready to be connected to a video game. We shall thus begin with the design of a set of Iceblox planning problems of increasing difficulty, in the spirit of the IPC: executable files for the planners, PDDL Iceblox problems files and scripts files to automate this Iceblox benchmarking process. If a planner fails these off line tests then it shall not be a good candidate for Iceblox video game playing.

What shall demonstrate that a planner is a good candidate for a connection to Iceblox? Solving the Iceblox benchmarks fast enough seems the obvious answer. Rather, the question should be: what is the time limit beyond which the current Iceblox situation is so dangerous that an action has to be taken right away, thus changing the initial state of the planning problem and consequently asking for re-planning? Iceblox has a good playability when the frame rate is about 30 frames per second, that is, when the flames coordinates (in pixel) are updated about 30 times per second. Luckily, the sprites are 30 by 30 pixels; it then takes about 1 second for a flame to move from one crossroad to another. According to the Iceblox code, a flame gets a random new direction between 1 and 4 crossroads, while keeping in mind that the new direction at the fourth crossroad might just be the same than the previous one. We can then consider that the limit is when Pixel Pete is 4 crossroads away from a flame, which gives a plan search runtime of at most 4 seconds. If a plan has been found then it needs to be executed, which undoubtedly takes time to trigger. Consequently, the flame should not reach the fourth crossroad and since movement between two crossroads is not interruptible, the flame should not reach the third crossroad before the plan search ends, which gives us a time limit of 3 seconds.

A planner shall be a good candidate for Iceblox video-game playing if it can (off line) solve Iceblox planning problems within the time limit of 3 seconds. This time limit sorts out dangerous flames from harmless flames.

4 Minimal Iceblox Situations

Three examples. We here describe three Iceblox planning problems of increasing difficulty which we designed in order to get a set of good candidate planners. Both plan length and planning problem size shall be used as criteria of difficulty: the larger number of predicates describing both the initial and the final states and the larger number of operators in the plan solution, the more difficult the problem.

It is necessary to collect a coin in each of the three problems. Figures 3, 4, 5 contain a screen shot of an iceblox planning problem and its PDDL code. See Figure 3 for the simplest of our three problem; this is obviously a "Welcome-to-Iceblox-world!" problem. In Figure 4, we introduce danger from one flame with only one weapon to kill this flame. In Figure 5, we set the case for two weapons to kill a flame guarding a coin. The properties of these three problems are the following:

See Figure	3	4	5
Requires flame fighting?	No	Yes	Yes
Number of weapons	0	1	2
Number of predicates (initial and final states)	7	10	13
Number of typed objects	4	8	10
Number of paths leading to the coin	1	1	2
Length of solution plan	2	4	4 or 5

Why these three problems? First because their number of predicates all are very small and yet they cover the basics of Iceblox game playing: these situations are so simple that if a planner fails at these problems then it is certainly not able to play iceblox. Moreover, although more problems have been designed, they confirm the results of Table 1. Given as Iceblox levels to an Iceblox planning system, these problems can be solved in real time; screen shots of the final Iceblox situations are gathered in Figure 6.

Planning problems predicates. The following predicates are used to describe both the initial and final states of the planning problems of Figures 3, 4 and 5 (crossroad$_{i,j}$ denotes the location at the intersection of line i and column j):

– (position i j): a sprite (Pixel Pete, ice block, iced coined, weapon) is at the crossroad$_{i,j}$.
– (extracted i j): the coin at the crossroad$_{i,j}$ has been collected by the player.
– (guard i_1 j_1 i_2 j_2): the flame at the crossroad$_{i_1,j_1}$ guards the coin at the crossroad$_{i_2,j_2}$. The idea of guarding a coin is linked to the 3 seconds time constraint (see the planners requirements): a flame makes dangerous the collection of a coin when it is within a range of 3 crossroads.
– (iced-coin i j): there is an ice block at the crossroad$_{i,j}$ which contains a coin.
– (protected-cell i j): there exists a path towards the crossroad$_{i,j}$. This path is safe: no flame makes this path dangerous.
– (reachable-cell i j): there exists a path towards the crossroad$_{i,j}$; there exists at least one flame putting this path in danger.
– (weapon i_1 j_1 i_2 j_2 i_3 j_3): there exists a weapon at the crossroad$_{i_2,j_2}$; Pixel Pete should push this weapon from the crossroad$_{i_1,j_1}$. The weapon shall stop sliding at the crossroad$_{i_3,j_3}$; this is useful information when an ice block needs more than one push before the final kick at a flame.

Operators use two more predicates:

– (blocked-path i j): there is an ice block on the path to crossroad$_{i,j}$.
– (blocked-by-weapon i_1 j_1 i_2 j_2): the weapon at crossroad$_{i_2,j_2}$ is on the path to crossroad$_{i_1,j_1}$.

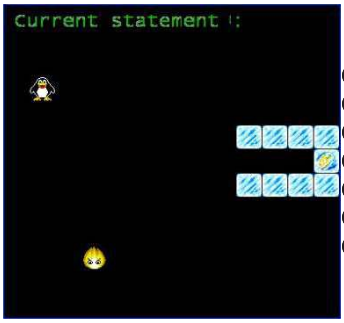
Over all the operators we designed, 4 proved to be critical for Iceblox game playing: move-to-crossroad, destroy-weapon, kick-to-kill-guard and extract. We hope their names are self explanatory. There are more (e.g. pushing a block to lock flames in a maze of ice blocks and rocks) but basic Iceblox playing is impossible if you don't get those 4 operators. Due to space limitations, we only give the PDDL code of kick-to-kill-guard and extract; these operators are given to the planners for benchmarking and playing:

```
(:action extract
    :parameters (?coinx - coord-i ?coiny - coord-j )
    :precondition (and (protected-cell ?coinx ?coiny)
            (iced-coin ?coinx ?coiny) (position ?coinx ?coiny)
            (reachable-cell ?coinx ?coiny))
    :effect (and (extracted ?coinx ?coiny)
            (not (iced-coin ?coinx ?coiny))
            (not (protected-cell ?coinx ?coiny))
            (not (reachable-cell ?coinx ?coiny))))

(:action kick-to-kill-guard
    :parameters
            (?reachablewx - coord-i ?reachablewy - coord-j
             ?weaponx - coord-i ?weapony - coord-j
             ?newweaponx - coord-i ?newweapony - coord-j
             ?guardx - coord-i ?guardy - coord-j
             ?coinx - coord-i ?coiny - coord-j
             ?blockedx - coord-i ?blockedy - coord-j)
    :precondition (and (iced-coin ?coinx ?coiny)
            (position ?reachablewx ?reachablewy)
            (guard ?guardx ?guardy ?coinx ?coiny)
            (weapon ?reachablewx ?reachablewy ?weaponx ?weapony
                ?newweaponx ?newweapony)
            (reachable-cell ?weaponx ?weapony)
            (protected-cell ?weaponx ?weapony))
    :effect (and (position ?weaponx ?weapony)
            (protected-cell ?coinx ?coiny)
            (blocked-by-weapon ?blockedx ?blockedy ?newweaponx ?newweapony)
            (reachable-cell ?newweaponx ?newweapony)
            (protected-cell ?newweaponx ?newweapony)
            (not (reachable-cell ?weaponx ?weapony))
            (not (protected-cell ?weaponx ?weapony))
            (not (reachable-cell ?blockedx ?blockedy))
            (not (guard ?guardx ?guardy ?coinx ?coiny))
            (not (weapon ?reachablewx ?reachablewy ?weaponx ?weapony
                ?newweaponx ?newweapony))))
```

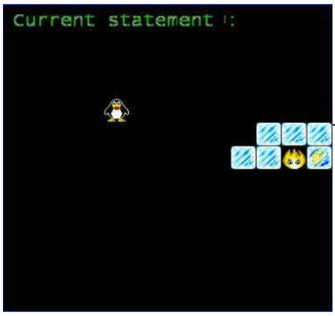
These operators can be much simpler and indeed we designed and used very simple versions of them. But of course, there is a compromise between simplicity which questions the overall utility of all this, and complexity which gets the planners nowhere. It is very easy to put more information in the operators than necessary; and then: bye bye runtimes!

Results. Several planners have been tested against these three problems (and others), with a 3.2 GHz Pentium 4 PC, loaded with 1Gb of memory. The runtimes, in seconds,



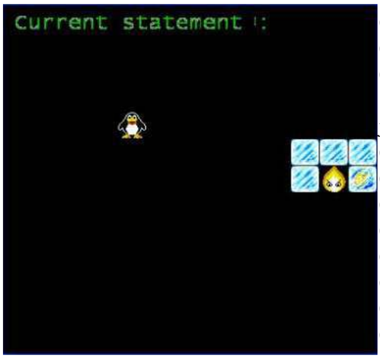
```
(define (problem first-case)
(:domain iceblox-strips)
(:objects i-1 - coord-i i-12 - coord-i j-1 - coord-j j-4 - coord-j)
(:init (position i-1 j-1) (iced-coin i-12 j-4)
(protected-cell i-1 j-1) (reachable-cell i-1 j-1)
(protected-cell i-12 j-4) (reachable-cell i-12 j-4))
(:goal (extracted i-12 j-4)))
```

Fig. 3. The danger is away, the path is obvious: you're fired if you can't solve this one



```
(define (problem second-case)
(:domain iceblox-strips)
(:objects i-4 - coord-i i-12 - coord-i i-11 - coord-i j-2 - coord-j
j-4 - coord-j j-3 - coord-j j-11 - coord-j j-10 - coord-j)
(:init (position i-4 j-2) (iced-coin i-12 j-4)
(protected-cell i-4 j-2) (reachable-cell i-4 j-2)
(guard i-11 j-4 i-12 j-4) (protected-cell i-11 j-3)
(weapon i-11 j-2 i-11 j-3 i-11 j-10 i-11 j-11)
(reachable-cell i-11 j-3) (reachable-cell i-12 j-4))
(:goal (extracted i-12 j-4)))
```

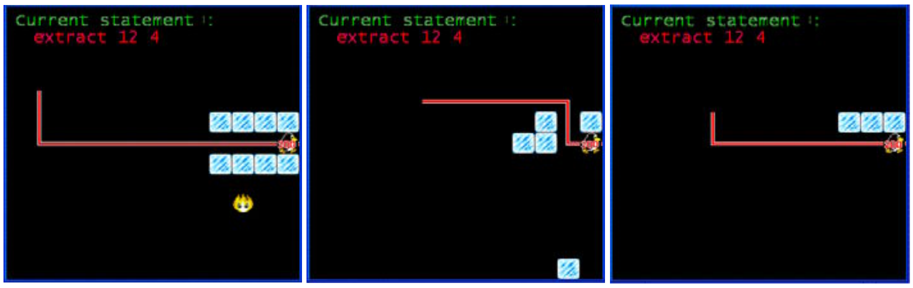
Fig. 4. There is danger, but the fight is easy



```
(define (problem third-case)
(:domain iceblox-strips)
(:objects i-4 - coord-i i-12 - coord-i i-11 - coord-i i-9 - coord-i i-
10 - coord-i j-2 - coord-j j-4 - coord-j j-3 - coord-j j-11 - coord-j
j-10 - coord-j)
(:init (position i-4 j-2) (iced-coin i-12 j-4)
(protected-cell i-4 j-2) (reachable-cell i-4 j-2)
(guard i-11 j-4 i-12 j-4) (protected-cell i-10 j-4)
(weapon i-9 j-4 i-10 j-4 i-11 j-4 i-12 j-4)
(weapon i-11 j-2 i-11 j-3 i-11 j-10 i-11 j-11)
(protected-cell i-11 j-3) (reachable-cell i-10 j-4)
(reachable-cell i-11 j-3) (reachable-cell i-12 j-4))
(:goal (extracted i-12 j-4)))
```

Fig. 5. Two unsafe paths: one shorter, one longer…

averaged over ten runs, are gathered in Table 1. The planners appearing in Table 1 reflect all the cases which happened during the tests: some planners rejected part or all of the PDDL files, reporting PDDL mistakes and some planners found no solution to



```
Current statement :
  extract 12 4
```

```
Current statement :
  extract 12 4
```

```
Current statement :
  extract 12 4
```

Fig. 6. When off line benchmarking becomes real time Iceblox playing. The red and white track marks the path followed by the penguin during the plan execution.

Table 1. Performances of several planners on PDDL files of the 3 problems of Figures 4, 5 and 6. A typed and untyped coordinates version was tested for each problem. All times, averaged over ten runs, are in seconds; planners were given 120 seconds to search for a solution although the required time limit is 3 seconds (see the requirements for planners).

	Typed			Untyped		
	Fig. 3	Fig. 4	Fig. 5	Fig. 3	Fig. 4	Fig. 5
FF [9]	0.01	0.49	1.83	0.01	> 120	
HSP2 [12]	PDDL Syntax errors					
Metric-FF [13]	0.01	0.33	1.07	0.01	> 120	
Qweak [10]	0.01	0.02	0.02	0.02	0.02	0.02
SGPlan [14]	0.01	0.35	1.14	0.01	> 120	
STAN [15]	0.18	PDDL Syntax errors		0.2	> 120	
TSGP [16]	0.03	25	> 120	0.05	> 120	
YAHSP [17]	0.01	No plan found		0.01	No plan found	

the problems. Hopefully, others correctly read the PDDL files; among those planners, some found solutions within the time limit and others did not. All the solutions found were correct: no planner we tested returned an incorrect plan.

Four planners, FF, Metric-FF, Qweak and SGPlan compose the set of good candidates for playing Iceblox. We eventually reduced the set to the well known and successful FF [9] and the less known and fastest but exotic Qweak [10][11].

Again, *no* planner was tweaked during both benchmarking and game playing (see the requirements for planners).

As you can notice, ice blocks in this design can only be pushed for killing flames guarding an iced coin and there is no action to crush an ice block. Indeed, we implemented the crushing of ice blocks in the execution module. Then, by carefully designing our levels, both FF and Qweak were able to play large Iceblox levels (i.e. 12 lines by 12 columns with at most 3 flames and several ice blocks) almost in real-time.

In the next section, we see how to plan the pushing and crushing of ice blocks with FF and Qweak.

5 Game Playing Situations

The previous game situations might be perfect for the selection of PDDL-based planners, but they are far from real iceblox game playing situations. In this section we carry on testing and see how more realistic game playing situations can be handled by the planners FF and Qweak. Looking at figure 1 we can observe that putting more objects in an Iceblox game level produces more realistic game situations: more ice blocks, more coins, more flames, more weapons,... We need problems with more objects, and consequently more predicates and more actions. However, FF's instantiation mechanism is sensible to more actions with more parameters. Consequently, if we propose more actions to the planners, we should be careful with the number of objects of our problems.

The first four problems we designed have the following properties:

See figure 7	*top-left*	*top-right*	*bottom-left*	*bottom-right*
Requires flame fighting?	No	Yes	Yes	Yes
Number of weapons	0	1	1	4
Number of predicates (initial and final states)	7	10	10	19
Number of typed objects	5	8	8	15
Number of paths leading to the coin	1	1	2	4
Length of solution plan	2	4	5	7

As you can observe, the size of these problems (except the *bottom-right*) correspond to the size of the problems of figures 3, 4 and 5. However, our domain file now has 7 actions instead of 4 and our new kick-to-kill-guard action now possesses 11 parameters while our push action has 6 parameters; the crush action, given below, only possesses 4 parameters.

New planning problem predicates are the following:

- (guard f i_1 j_1 i_2 j_2): the flame f at the crossroad$_{i_1,j_1}$ guards the coin at the crossroad$_{i_2,j_2}$. This predicate now possesses a new parameter f to record a flame number to facilitate the execution process: we may have already killed this flame, thus making needless to focus on it any longer; or it is another flame which is now dangerous and a new plan must be computed.
- (blocked-by-cell i_1 j_1 i_2 j_2): there is an ice block at the crossroad$_{i_1,j_1}$ which can be either pushed or else crushed when at the crossroad$_{i_2,j_2}$; this predicate is a rewriting of the two predicates (blocked-path i j) and (blocked-by-weapon i_1 j_1 i_2 j_2).
- (crushable i j): the ice block at the crossroad$_{i,j}$ can be crushed so that (reachable-cell i j) becomes true.
- (pushable i_1 j_1 i_2 j_2): the ice block at the crossroad$_{i_1,j_1}$ can be pushed; if pushed, it shall stop at the crossroad$_{i_2,j_2}$.

One operator use one more predicate:

- (connected-with-cell i_1 j_1 i_2 j_2): this new predicate is always used in conjunction with the predicate (reachable-cell i_2 j_2) thus asserting that the cell at the crossroad$_{i_1,j_1}$ also is reachable (see the set-reachable action below). In fact, any crossroad is reachable in the Iceblox game since it is always possible to push or crush an ice block to reach it. So, in theory, we should generate this predicate for many interesting positions we could detect and in particular around weapons. However, the dynamics of the game render this generation useless: flames move and weapons change.

We first implemented ice block crushing in the execution module (see next section) but we soon realized it was not coherent with our design of PDDL actions: crushing an ice block is a 7 push action in the same direction. Consequently, we designed an abstract push action gathering several basic pushes as our move action gathers several basic moves:

```
(:action crush-iceblock
    :parameters
        (?crushablex - coord-i ?crushabley - coord-j
         ?blockedx - coord-i ?blockedy - coord-j)
    :precondition (and (position ?crushablex ?crushabley)
        (reachable-cell ?crushablex ?crushabley)
        (protected-cell ?crushablex ?crushabley))
        (crushable-cell ?crushablex ?crushabley)
        (blocked-by-cell ?blockedx ?blockedy ?crushablex ?crushabley))
    :effect (and (reachable-cell ?blockedx ?blockedy)
        (not (reachable-cell ?crushablex ?crushabley))
        (not (crushable-cell ?crushablex ?crushabley))))
```

As discussed for the new predicate connected-with-cell, we designed an action to compute, through planning, the transitive closure of reachable-cell predicate. In practice, solution plans are rarely generated with this action.

```
(:action set-reachable
    :parameters
        (?cellx - coord-i ?celly - coord-j ?cellu - coord-i ?cellv - coord-j)
    :precondition (and (reachable-cell ?cellx ?celly)
        (connected-with-cell ?cellu ?cellv ?cellx ?celly))
    :effect (and (reachable-cell ?cellu ?cellv))))
```

Results. We tested both FF and Qweak against typed versions of the four problems of figure 7; anticipating bad runtimes, we traded our previous every-office-of-yesterday-machine to a newer 2.20GHz T7500 core duo loaded with 3Gb of memory. FF solves the first two problems in, respectively, 30 milli-seconds and 200ms, and fails to solve the last two, running out of memory: as predicted, FF's instantiation mechanism can be blamed for these failures. We thus decided to test Metric-FF and SGPLan as well, but with no more success than FF. Qweak solves the first three problems at about the same speed of 30ms and solves the last problem in 140ms: a faster machine entails faster runtimes.

Again, no planner was tweaked during these tests, but we did correct a couple of bugs in Qweak which appeared when testing even bigger problems.

Realistic iceblox levels can be solved in real-time. However, to achieve good playability in the video-game domain, everything must be fast: the construction of plans, of

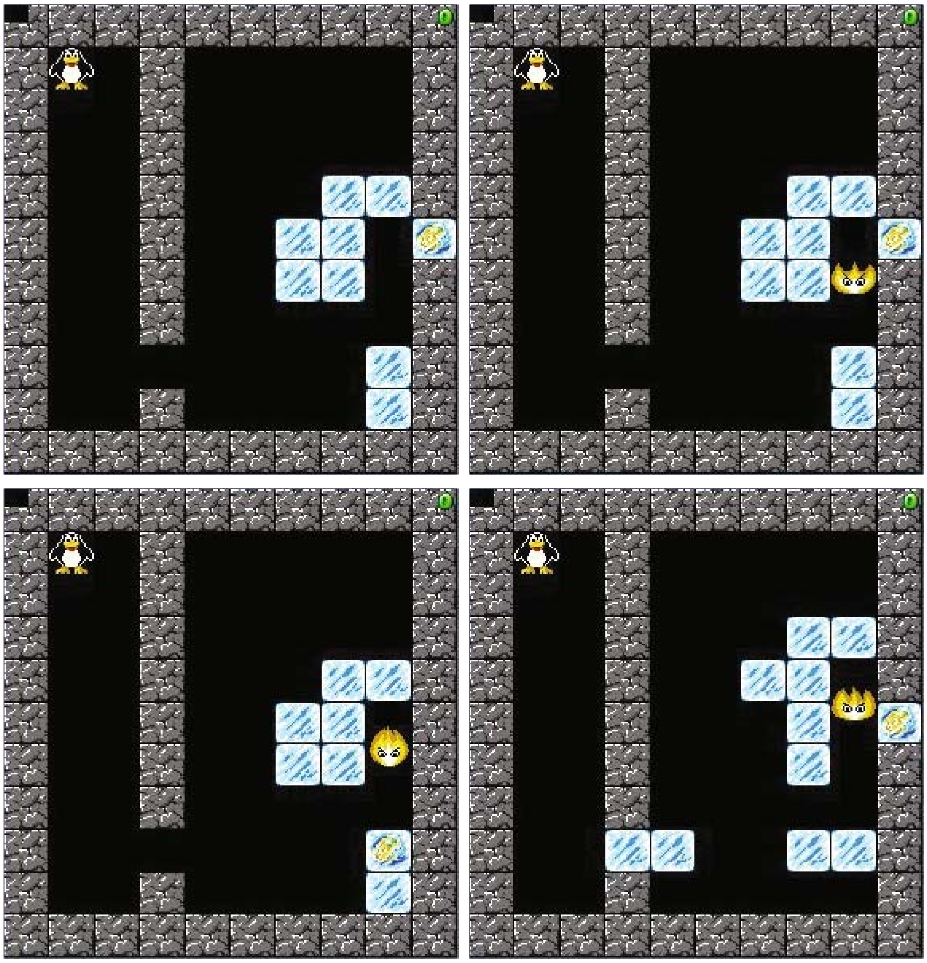


Fig. 7. The Penguin starts in line 1 and column 1 and its objective is to extract the only coin in each problem. When pushed, an ice block can slide only when its other side is free; an ice block can be crushed if its opposite side is not free. A flame dies when it collides with a sliding ice block which is then called a weapon. Because there even is no flame at all, the *top-left* problem is simpler than that of figure 3 and the *top-right* problem is similar to that of figure 4. The *lower-left* problem corresponds to that of figure 5: when pushed to kill the flame, the ice block in line 4 and column 8 shall stop just above the iced coin thus blocking the path to the iced coin; then, the Penguin must either walk round the ice blocks to reach the left side of the iced coin or else destroy the weapon which stopped sliding on the upper side of the iced coin. The *lower-right* problem appears to be more difficult than all other previous problems: first, there are four weapons which, once pushed, could collide with the flame; second, there is an obstacle on the path to these weapons: the entry point between the two rooms is blocked and an ice block must be crushed and another one must be pushed. Now, go to figure 8 to see how difficult these problems really are.

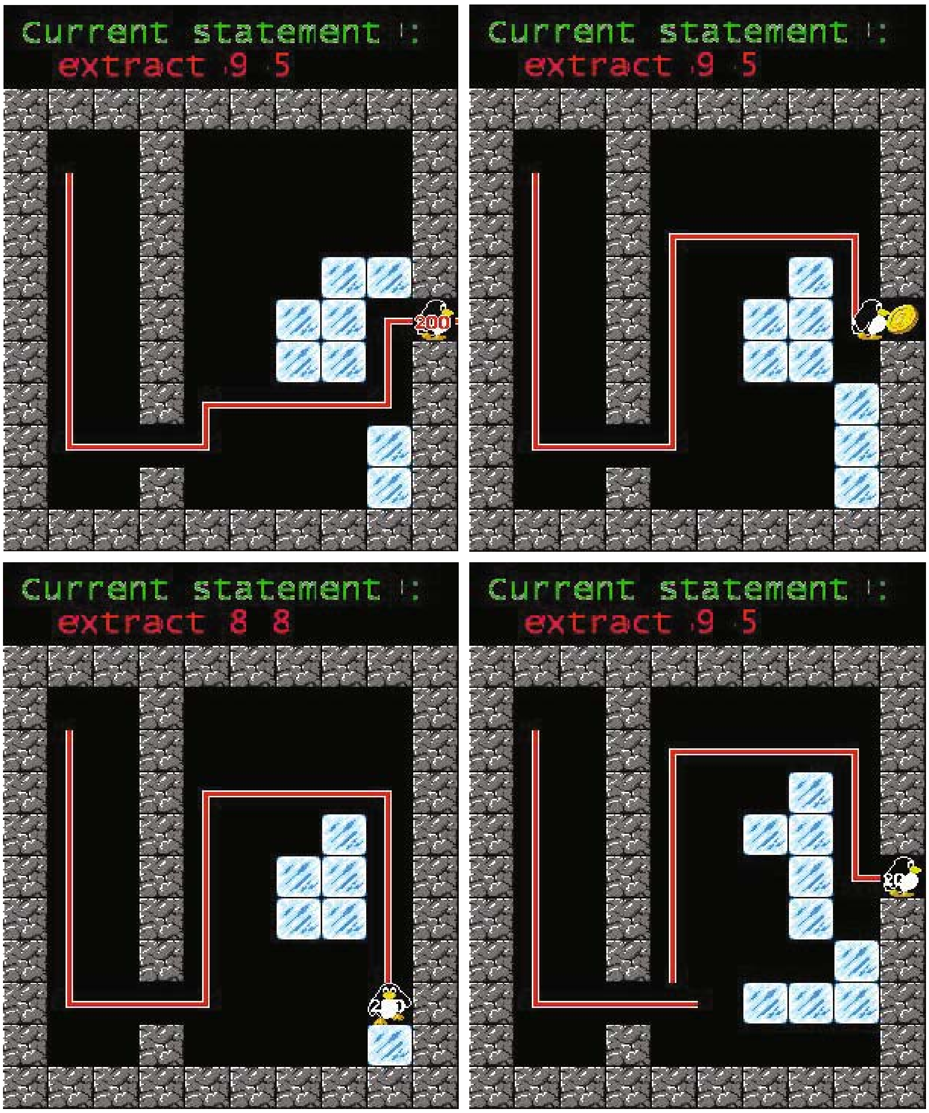


Fig. 8. The moment of truth for the planning problems described in figure 7. The four problems must be solved so we feed the planner with actions to push and crush ice blocks: 7 actions and 12 predicates in the domain file are now necessary to solve these problems whereas 4 actions and 10 predicates were needed for the problem of figures 3, 4 and 5. Current IPC planners can solve the two problems of the *top* line on today's machines but only when objects are typed and the `:typing` requirement appears in the PDDL problem file (several milliseconds for FF [9] on the *top-left* problem and 0.2s for the *top-right* problem). However, the two problems on the *bottom* line are too difficult: all planners but Qweak [10] fails to find a solution within 120 seconds: the 11 parameters of the kick-to-kill-guard action, the 6 parameters of the push action and the number of weapons in the *lower-right* problem create too many search alternatives for the planners. The red and white track marks the path followed by the penguin during the plan execution.

course, but also the generation of PDDL problems and the execution of plans and probably above all, the coupling of all these procedures to the game must not slow down the game. The next section digs our game playing architecture and, in particular, the on line generation of PDDL problems.

6 Game Playing Architecture

This section outlines the plan execution procedure and the generation of PDDL planning problem from Iceblox game situations. Finally we discuss path planning in the Iceblox domain.

6.1 The Plan Execution Process

Table 2 outlines the plan execution process; it is implemented as a thread and can give orders to the game through a global variable which usually contains the key just pressed by the user.

The tricky part is the compiling of the PDDL actions of the plan returned by the PDDL-based planner into basic instructions (the keys pressed by the user) for the game. This allows a regular sensing of danger and gives regular opportunities to response to unexpected events.

6.2 The Generation of PDDL Planning Problems

Table 3 presents how the collection-of-a-single-coin is generated as a PDDL planning problem during game playing. This very fast procedure, also implemented as a thread,

Table 2. The plan execution thread

```
Repeat
  If the first operator of the plan is Move-To-Crossroad($i$,$j$) then
    Compute a path from current location to crossroad$_{i,j}$
    Repeat
      Execute one basic move following that path
    Until crossroad$_{i,j}$ is reached
  Else
    If the ice block at crossroad$_{i,j}$ is not destroyed
      Execute a push action in direction of crossroad$_{i,j}$
    Else
      Execute a basic move towards crossroad$_{i,j}$
    End if
  End if
  Delete the first operator of the plan

  When the flame is no longer dangerous
    Ignore it
  When the flame is no longer aligned with the weapon or
      an unexpected flame becomes dangerous
    Find a weapon and use it
  When the penguin must leave the computed path
    Remember the current crossroad$_{i,j}$
    Get the penguin back to crossroad$_{i,j}$ as soon as possible
Until the plan is empty
```

Table 3. The PDDL Planning problem generation thread for the collection of a single coin

```
Document-the-path-to(object)
  Compute a path for this object
  Generate (reachable-cell i-object j-object)
  If this path is clear then
    Generate (protected-cell i-object j-object)
  Else
    For each ice block on this path do
      Generate (blocked-by-cell i-object j-object i-block j-block)
      If this ice block can be moved with a simple push then
        Generate (pushable-cell i-block j-block i-stop j-stop)
      Else   /* this ice block cannot move and must be crushed */
      Generate (crushable-cell i-block j-block i-crush j-crush)
      End if
    End for each
  End if
End Document-the-path-to

Generate PDDL planning problem header
Generate PDDL initial state header
Generate (position i-penguin j-penguin)
Find the (nearest or possibly flame-less) iced coin
Generate (iced-coin i-coin j-coin)
Document-the-path-to(iced-coin)
For each flame do
  If this flame is less than four crossroads away from the (iced-coin i-coin j-coin) then
    Generate (guard flame i-flame j-flame i-coin j-coin)
    For each nearest weapon in each of the four direction do
      Generate (weapon i-push j-push i-weapon j-weapon i-stop j-stop)
      Document-the-path-to(weapon)
    End for each
  End if
End for
Generate PDDL final state header and (extracted i-coin j-coin)
```

first computes what to do (either push or else crush) with ice blocks on its path to an iced coin. Then it finds weapons by looking in the four directions around a dangerous flame while managing the ice blocks it encounters on its path to these weapons.

6.3 Path Planning

Path planning is obviously used to compute paths from a location to another. This path planning procedure is first called during the generation of PDDL planning problem (see table 3) and then during the plan execution process (see table 2): path planning is an essential activity.

We implemented the A* [20] algorithm with ideas from [21]: for instance, we added a penalty for each direction change to the cost of a path in order to get more rectilinear paths, following our requirements on actions of kind 3 (see section 3 above). However, Iceblox is not a free path world and destructive operations must happen to find paths: searching on empty crossroads is not sufficient to find paths. Pushing and crushing ice blocks often are necessary actions to find paths, but both modify the game level in an irreversible manner: for the sake of completeness of the A* algorithm, modified game levels must be saved as new search spaces for path planning, which obviously costs memory and therefore time.

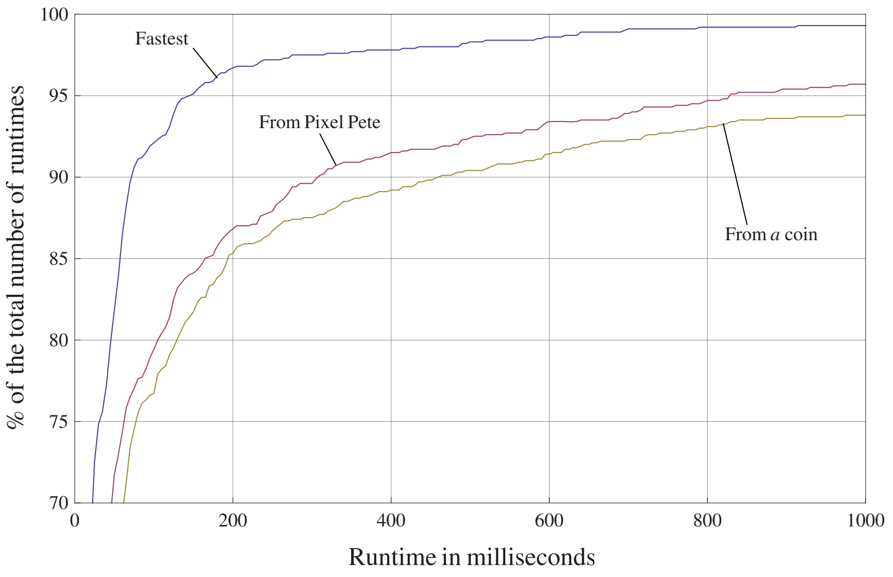


Fig. 9. On the horizontal axis are runtimes (milliseconds) from our implementation of A* which computes paths in Iceblox randomly generated levels (10 lines by 12 columns); Pixel Pete appears in the top left corner, and 9 rocks, 5 coins and 35 pushable ice blocks are randomly placed in this order on the remaining cells. This makes an amazing 10 342 621 660 587 151 106 372 657 654 037 758 770 942 528 117 680 different game levels (see section 6.3). The cost of a path is the cost of actions (this cost is 1 for a basic move in each of the four directions, 1 for pushing an ice block and 7 for crushing an ice block) plus the number of corners (i.e. direction changes) plus the manhattan distance to the goal position. 200 levels were randomly generated; we recorded both the runtime to compute a path from one coin to Pixel Pete (*the lowest curve*) and from Pixel Pete to the same coin (*the second curve*); the five coins of each level were tested. Consequently, 1000 paths have been computed in each of the two directions. How fast (or slow?) are these computations? As we can observe in the above figure, 85% of the computations from a coin took less than 2OO milli-seconds whereas 87% of the computations from Pixel Pete took less than 200ms; so should we prefer to compute a path always from Pixel Pete? The answer is no: the computations from Pixel Pete produced 513 of the fastest runtimes while the computations from a coin produced 547 of the fastest runtimes (60 tests produced the same runtime). But despite these 547 fastest runtimes, *the lowest curve* in the above figure shows that it generally takes a little more time to compute a path from a coin than from Pixel Pete. However, if we choose the fastest runtime of the two available runtimes for each coin, it appears that 97% of the computations took less than 200ms which is just fine (*the highest curve*).

The figure 9 shows the proportions (vertical axis) of runtimes (horizontal axis) in milli-seconds above a given runtime for paths computed from Pixel Pete to a coin (*the middle curve*) and from a coin to Pixel Pete (*the lowest curve*). For instance, 95% of the runtimes are below 800ms, which clearly is unsatisfactory for real-time playability. If we look for path planning runtimes equivalent to PDDL-based plan construction runtimes, we observe that around 86% of the runtimes are below 200ms (which is not very fast yet). 1000 paths (in both ways: 2000 paths in total) have been computed over 200 generated levels where Pixel Pete appears in line 1 and column 1, 9 blocks, 5 coins and 35 ice blocks are placed at random on the remaining cells, in this order. The initial position of the penguin is fixed so we first have to choose 9 cells among the 119 remaining cells to place the rocks, then choose 5 cells among the 110 remainings cells and finally choose 35 cells among the 105 remaining cells; let lc be number of lines times the number of columns, we have:

$$\binom{(lc-1)}{9} \times \binom{(lc-10)}{5} \times \binom{(lc-15)}{35} = \frac{(lc-1)!}{5!\,9!\,35!\,(lc-50)!}$$

This total number of such levels has to do with astronomy and we leave the computations to the numbers lovers; it is, however, not difficult to remark that 200 generated levels is a desperately small number compared to the numbers provided by the above formula. However, it seems that more tests only reinforce the observed results of figure 9.

By taking the minimum value of the two available runtimes for each coin, we get the *highest-curve* in figure 9, where 97% of these minimum runtimes are less than 200 ms. Consequently, we are left with several options: (i) let the two computations (from Pixel Pete and from a coin) run concurrently and take the first returned path, (ii) search in a destructive manner (whatever the path), (iii) investigate a bi-directional search procedure, (iv) spend some time optimizing any step of our A* implementation, in the spirit of [22] and (v) focus on something else. After failing to gain much with the third and fourth options, perhaps strangely, we chose the fifth option: it happens that, finally, the gaming experience is not much affected by a somewhat slow but nicely correctly implemented path planning procedure. Now, we advise the wise programmer to first investigate option (ii) before trying option (i).

7 Conclusions

This paper reports on the design and implementation of a PDDL-based planning system (problem generator, planner and plan execution) which, most of the time, is able to play an arcade game in real-time. We detailed engineering decisions because this research project showed us that an efficient planner itself (PDDL-based or not) certainly is not sufficient to achieve real-time playability. In the case of an arcade game, everything must be fast; or, if preferred, nothing must be slow. To avoid slowing the game, many planning knowledge oriented decisions must be taken: which predicates must be designed to represent a gaming situation? What is the balance between detailed planning operators and a clever plan execution system? What can be put in a thread? What part of planning computations can be distributed over multiple graphic frames? This paper proposes answers to these questions, mainly from an engineering perspective and not from a theoretical perspective: decisions have, most of the time, been made in favour of the game playability.

Due to the good quality of the path planning computation, Pixel Pete looks pretty rational at the speed of the flames. Sincerely, we thought many times we'd never make it and the granularity of our Plans is crucial to achieve our goal: we just produce Planning problems with a small number of predicates. Since then, we have reused our architecture in the commercial serious game domain; we successfully coupled our planning system to Virtual Battle Space 2 [23] [24], in a bottom-up manner. We are actually working on extending this coupling.

Acknowledgements. This work is part of a 3 year project founded by the Fondation Saint-Cyr. Thanks to Jeremy Buisson for his help on multithreading. Thanks to Rick Alterman, Marc Cavazza, Jon Gratch, David Kirsh, Carlos Linarès, Jeff Orkin and Adrian Smith for helpful discussions since the beginning of this project.

References

1. O'Brien, J.: A flexible goal-based planning architecture, pp. 375–383. Charles River Media (2002)
2. Orkin, J.: Applying goal-oriented action planning to games. In: Rabin, S. (ed.) AI Game Programming Wisdom 2, pp. 217–227. Charles River Media (2003)
3. International Planning Competition (1998–2009), `http://ipc.icaps-conference.org/`
4. Gerevini, A., Haslum, P., Long, D., Saetti, A., Dimopoulos, Y.: Deterministic planning in the fifth international planning competition: Pddl3 and experimental evaluation of the planners. AI Journal 173, 619–668 (2009)
5. Hornell, K.: Iceblox (1996), `http://www.javaonthebrain.com/java/iceblox/`
6. Bartlett, N., Simkin, S., Stranc, C.: Java Game Programming. Coriolis Group Books (1996)
7. Wikipedia: Pengo, arcade game (2007), `http://www.wikipedia.org/`
8. Orkin, J.: Three States and a Plan: The A.I. of F.E.A.R. In: Proceedings of the Game Develeopper Conference, 17 pages (2006)
9. Hoffmann, J.: FF: The Fast-Forward planning system. AI Magazine 22(3), 57–62 (2001)
10. Bartheye, O., Jacopin, É.: New results for arithmetic constraints partial order planning. In: 24th Workshop of the UK Planning and Scheduling Special Interest Group, London, UK, December 15-16 (2005)
11. Nareyek, A., Fourer, R., Giunchiglia, E., Goldman, R., Kautz, H., Rintanen, J., Tate, A.: Constraints and AI Planning. IEEE Intelligent Systems, 62–72 (March/April 2005)
12. Bonet, B., Geffner, H.: Heuristic search planner 2.0. AI Magazine 22(3), 77–80 (2001)
13. Hoffman, J.: The metric-ff planning system: Translating "ignoring delete list" to numeric state variables. Journal of AI Research 20, 291–341 (2003)
14. Chen, Y., Wah, B., Hsu, C.W.: Temporal planning using subgoal partioning and resolution in sgplan. Journal of AI Research 26, 323–369 (2006)
15. Long, D., Fox, M.: Efficient implementation of the plan graph in stan. Journal of AI Research 10, 87–115 (1999)
16. Coles, A., Fox, M., Long, D., Smith, A.: Planning with respect to an existing schedule of events. In: Proceedings of the 17th ICAPS. AAAI Press, Menlo Park (2007)
17. Vidal, V.: A lookahead strategy for heuristic search planning. In: Proceedings of the 14th ICAPS, pp. 150–159. AAAI Press, Menlo Park (2004)
18. Bartheye, O., Jacopin, É.: Planning as a software component: a report from the trenches. In: 26th Workshop of the UK Planning and Scheduling Special Interest Group, Prague, CZ, December 17-18 (2007)
19. Microsoft: Directx 9 (2009), `http://msdn.microsoft.com/en-us/directx`

20. Pearl, J.: Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley, Reading (1984)
21. Rabin, S.: A* aesthetic optimizations. In: Deloura, M. (ed.) Game Programming Gems, pp. 264–271. Charles River Media (2000)
22. Rabin, S.: A* speed optimizations. In: Deloura, M. (ed.) Game Programming Gems, pp. 272–287. Charles River Media (2000)
23. Bartheye, O., Jacopin, É.: A planning plug-in for virtual battle space2: A report from the trenches. In: Proceedings of the Spring Simulation Multiconference, 4 pages (2009)
24. Bartheye, O., Jacopin, É.: A real-time pddl-based planning component for video games. In: Proceedings of 5th AIIDE. AAAI Press, Menlo Park (to appear, 2009)

Agent-Based Aircraft Control Strategies in a Simulated Environment

Daniel Castro Silva, Ricardo Silva, Luís Paulo Reis, and Eugénio Oliveira

Faculty of Engineering of the University of Porto – Department of Informatics Engineering / Artificial Intelligence and Computer Science Laboratory, Rua Dr. Roberto Frias s/n 4200-465 Porto, Portugal
{dcs, ee99208, lpreis, eco}@fe.up.pt

Abstract. In the recent years, game engines have been increasingly used as a basis for agent-based applications and multi-agent systems, alongside traditional, more dedicated, simulation platforms. In this paper, we test and compare different control strategies for specified high-level maneuvers in aircraft within a gaming simulation environment, each aircraft represented by an independent agent, focusing on communication necessities and maneuver effectiveness. An overview of the simulation environment's capabilities, with focus on the structured experiences system, shows the potentials of this platform as a basis for the more comprehensive goals of our project, allowing for the definition and execution of cooperative missions, such as surveillance and search & rescue operations.

1 Introduction

Simulation tools are widely used in several fields of research, providing researchers with the means to develop their work in a cost- and time-effective manner. These and other advantages of simulation environments have led to a significant growth in their general use by the scientific community, as well as in several business areas, supporting decision-making processes, planning, training, and many other tasks. Alongside professional simulation tools, designed specifically to the research purpose they are used in, game engines and gaming tools are being adopted more and more by the scientific community as serious tools in their work. Serious gaming explores the use of games and game-like applications in training or learning, or using them as decision support tools, to improve business processes. Not only the entertainment organizations use of these new applications, but also governmental and military organizations, mainly for training purposes, be it in the leadership and management areas, or in a more technical approach, using primarily simulation tools [1].

One field of research where simulator tools are extensively used is aviation. In fact, simulators used in this area range from multi-million dollar full hardware and software simulators used to train professional pilots to freely available flight simulators, which have been used mainly for entertainment purposes and by aviation enthusiasts. As an example, one of the world's most famous simulators, NASA's VMS (Vertical Motion Simulator), is a full simulator capable of six degrees-of-freedom movements, housed in a ten-story building in Ames Research Center, California [2][3]. This simulator is capable of providing different simulation experiences, thanks to the ICAB (Interchangeable Cab) feature, which allows for a modular interior of the simulation cabin, presenting the pilot with different cockpit interfaces. An impressive 1300 variables can be retrieved from the simulator, allowing for thorough data analysis of simulation sessions. This simulator has been used for pilot and astronaut training but also in cooperation with other entities [4]. On the other end, flight simulators such as FlightGear (an open-source flight simulator[1]) are available for download at no cost or at a reduced price. Some of these low-cost flight simulators have, in the latest years, achieved a high level of realism, both in the simulation of aircraft systems, kinematics and weather conditions, and also in the visualization of all these aspects with a realistic and detailed terrain. Some of them have been certified by the FAA (Federal Aviation Administration) for pilot training, when used in conjunction with the proper hardware simulator. For instance, the X-Plane simulator[2] is used by some simulator companies, such as Simtrain as the software basis for simulation [5].

In the recent years, simulators in several areas have evolved into more complex systems, and in many cases a multi-agent system approach has been adopted. In these systems, the participating entities are represented by independent agents, which in turn communicate with each other, coordinating information and actions. In the aeronautical area, this approach has also been adopted, usually with each agent representing one aircraft (or even several agents representing the several systems of one single aircraft, if a more detailed simulation of a specific system is required). Such systems are used to develop new strategies to solve problems such as collision avoidance among UAVs (Unmanned Aerial Vehicles) [6], formation flight or fault tolerance [7], among others. These systems should also address the interaction with human beings, allowing for a real-time awareness of the situation and the possibility to define missions, or even to override decisions made by the agents [8].

The authors' vision is to make use of such a system in order to develop strategies to use in cooperative missions, such as surveillance and search & rescue scenarios. This paper focuses on control strategies for aircraft within the simulation environment, and their effectiveness in terms of both communication necessities and maneuver smoothness and accuracy. Three distinct strategies were tested to control different aircraft, and the experimental results were analyzed as to assess which of the methods is most suited to use in the simulator. As to provide with a more comprehensive context on the project, the following paragraphs summarize the steps that were previously taken.

The main goals of the overall project are to use a simulation platform as a basis for the design and cooperative execution of joint missions. These missions are to be performed by a team of heterogeneous vehicles, including planes, helicopters, land vehicles, and submersible vehicles, capable of communicating with

[1] More information available online at `http://www.flightgear.org/`
[2] More information available online at `http://www.x-plane.com/`

one another, in order to coordinate their actions. The applications of such a system are diverse, including surveillance (forest surveillance that provides an early fire detection system; coastal and border patrol, in order to detect and track illegal activities, such as smuggling; urban observation that would detect dense traffic patterns, preventing larger traffic jams; and many other applications), reconnaissance and target tracking (especially useful in military operations and law enforcement activities, to provide real-time valuable information about enemy movements, or to follow a fugitive until apprehended by competent entities), aiding in search & rescue operations, and many other applications.

The project's envisioned architecture incorporates the chosen simulator, a number of external agents, each representing one vehicle within the team, as well as some valuable peripheral services, such as a monitoring application, capable of real-time vehicle tracking, as briefly mentioned in the results section, simulation logging that enables mission replay and further analysis, and an interface with external modules, for interaction with real vehicles.

1.1 Simulation Engine

Previous work on this project included a thorough study of simulation platforms, which resulted in the choice of Microsoft Flight Simulator X (FSX)[3] as the platform to be used. This was based on the platform's admirable graphical features, simulation of aircraft dynamics and interaction with the environment, possibility to inject failures in certain aircraft systems and sensors, excellent application programming interface documentation, with hundreds of simulation variables that can be read and written to, hundreds of events that can be sent to the simulator, and many other features (which allow an external application to interact with the simulator in real-time), as well as the available mission system, with its outstanding possibilities [9].

FSX's programming interface, SimConnect, provides a flexible, powerful and robust client-server communications protocol that allows asynchronous access to hundreds of simulation variables and events. Some of these variables can only be read, but nearly two hundred can also be written to; several hundred events can be sent to the simulator, and there are several dozens of functions to deal with the weather system, missions, in-game menus, AI objects, communication and data access and manipulation, among other useful features.

Structured experiences, or missions, as they are commonly known, are a feature of FSX that allows regular users to have a different, interactive experience of flight, with measurable goals. FSX includes numerous missions, ranging from tutorials that teach the user how to fly an aircraft, to racing missions, simulating the Red Bull Air Race environment, as well as several transport or rescue operations, among others. The mission system allows for the definition of objects, areas, triggers and actions that can be linked to work together in an orchestrated manner, producing diverse, realistic and complex missions [10].

Microsoft has also already recognized the advantages of simulation in various business areas and the potential of these structured experiences, and is

[3] More information available online at `http://www.fsinsider.com/`

commercializing the engine behind FSX as a new enterprise-oriented product, called ESP [11]. There are numerous applications of this technology, and several high-profile companies have already signed partnerships with Microsoft for the use of ESP, such as FlightSafety International, SAIC, Lockheed Martin and Northrop Grumman [11] [12]. These companies, as well as others that use this product, will be able to use it to train staff (not only in piloting skills, but also management skills for airline companies and airports, for instance), learning, research or decision support – by simulating different scenarios, the most favorable one can be identified, problems and design flaws can be corrected, business processes can be improved.

1.2 Autopilot Systems

Although autopilot systems are usually associated with aircraft, they can be used in any kind of vehicle, including boats, cars, or even missiles. In this paper, however, the focus is on aircraft autopilot systems. There is a variety of autopilots commercially available for small unmanned aircraft, usually comprised of light-weight hardware to connect to the aircraft, and a control station, also often including some software to interact with the system. These autopilots range from simple one-axis controllers to full three-axis systems, including redundant processors, sensor diagnostics and failure tolerance, medium- to long-range communication system and many other useful features when dealing with payloads [13]. Integration and testing of autopilot systems with the aircraft has to be carefully executed, as to calibrate the system to the aircraft in question [14]. Commercial aircraft also feature autopilot systems, which can control the aircraft from takeoff to land without human intervention (these systems are, however, usually only used during the leveled part of the flight, and during landing, when visibility conditions are below certain limits). These full autopilot systems use redundant computers to ensure that control decisions are correct, and provide a more stable flight than human pilots would, lowering fuel consumption at the same time.

The rest of this paper is organized as follows. In the next section, the three approaches that were compared are explained in more detail. Section three presents the settings for the experiments that were conducted and section four presents the results that were attained, comparing the approaches. Finally, in the last section, some conclusions are withdrawn and lines of future work are presented.

2 Control Strategies

In this section, the three devised control strategies are presented and briefly compared. The first approach consists on direct manipulation of the aircraft controls, such as aileron and elevators through a PID controller. The two other approaches use autopilot systems. The second one makes use of the autopilot system available to actual pilots, delegating the handling of aircraft controls to the system. The third approach uses the AI autopilot, used by the simulator to perform automated flights. The differences between these methods and the specifics of each one are detailed in the following subsections.

2.1 PID Controller

The first approach to controlling an aircraft within the simulation environment is to recreate the actions of a pilot when attempting to follow a certain route. This was accomplished by direct manipulation of the main controls of the aircraft, namely the throttle, aileron and elevator. The throttle control is used to control the speed of the aircraft. The elevator control is used to control the pitch angle of the aircraft (sometimes referred to as angle of attack, pitch measures the nose up or down angle of an aircraft), and, when used in conjunction with the throttle control, it allows to control the altitude of the aircraft – in order to climb, the elevator must be set to a positive value, and the throttle should be set to full throttle; in order to descend, the elevator must be set to a negative value, and the throttle should be set to idle. The aileron control is used to control the bank angle of the aircraft (also called roll, it is the angle of rotation of the plane about its longitudinal axis). In order to make a turn, the aircraft rolls toward the inner part of the desired turn. By using these controls together, higher-level maneuvers are executed.

In order to handle these aircraft controls, a PID (Proportional-Integral-Derivative) controller was used. A PID controller is a generic control loop feedback mechanism that attempts to correct the error between a measured variable and the desired value by calculating a corrective action that can adjust the process accordingly. It is a well-known controller from the control theory field, and has been used successfully in the industry for many years. Its numeric implementation consists on the evaluation of (1), where e is the difference between the reference value and the feedback measured value, τ is the time in the past contributing to the integral response and K_p, K_i and K_d are the proportional, integral and derivative gains, respectively. The proportional term adjusts the output signal in direct proportion to the error, the integral term is proportional to both magnitude and duration of the error, and the derivative term measures the approximate rate of change of the error. Tuning the gain factors of the PID controller is an important step, to assure optimal values for the desired control response [15].

$$output\left(t\right) = k_p e\left(t\right) + k_i \int_0^t e\left(\tau\right) d\tau + k_d \frac{de}{dt} \quad . \tag{1}$$

For this approach, the control agent communicates with the simulated aircraft at a given rate, in order to send the current values, calculated with the PID controller. Ideally, the communication would be uninterrupted, since continuous adjustments to the aircraft controls should be made in order to maintain the desired flights settings. However, the simulator only sends and accepts values once per simulation cycle, and to send data at a higher rate would be a waste of processing time. Moreover, one has to account for communication latencies that make it difficult to receive data from a given simulation cycle and to send the corrective values that reach the simulator in time for the following cycle.

There is a simulation variable that is modified by the control agent for each of the three aircraft controls that are manipulated by the controller, and other variables that can be read in order to assess the current position, speed and attitude of the aircraft, used in the calculations as mentioned above.

In order to make a banked turn with a given radius (for circling maneuvers, for instance), some additional information has to be taken into account. The radius of a banked turn is given by (2), where v is the true airspeed of the aircraft, g is the acceleration due to gravity and θ is the bank angle of the aircraft.

$$R = \frac{v^2}{g \tan \theta} \quad . \tag{2}$$

Given (2), one can determine the necessary bank angle to generate a banked turn with a given radius at a given speed.

2.2 Autopilot

In this approach, the autopilot features of aircraft are used. Autopilot systems assume various forms, ranging from simple wing-levelers to complete systems, capable of controlling an aircraft from takeoff to land. Most modern aircraft feature altitude, heading and speed controls, allowing the pilot to set the desired values for each feature. The altitude control manipulates the elevator and throttle (if auto-throttle is available) in order to reach and maintain a certain altitude. The heading control manipulates the aileron in order to make a banked turn until the desired heading is reached, and the speed control automatically adjusts the throttle to maintain the desired airspeed. This approach uses the available autopilot systems as a pilot would, to adjust the aircraft's course to the desired settings.

There is a straightforward limitation to this approach, which is the necessity for the presence of an autopilot system. In fact, many smaller, older aircraft do not possess an autopilot system, or it is only comprised of the already mentioned wing-levelers, capable only of maintaining a straight level flight (but unable to change altitude, heading or speed). The authors, however, feel that this limitation does not impose a major problem, since there are several commercially available autopilot systems for small unmanned aircraft, such as the Piccolo autopilot system, from CloudCap Technology [16], which are usually used for the type of aircraft that are intended to be used in real-life experiments.

With this approach, there are limits that need to be taken into account when planning the maneuvering scenarios, such as the autopilot system limits for the bank angle, for instance. As previously seen, the bank of a plane influences the radius of the curve, and as such, a minimum radius for turns is calculated based on the aircraft cruise speed and maximum bank angle at the beginning of the simulation, and that minimum value is enforced on maneuvers that require the plane to move in a circular fashion.

Since these systems do not allow for a direct control of the bank angle (some systems have the capability to further limit the maximum bank angle to usually half of that value, to produce smoother flights), a circular path is a bit more complicated to achieve. For circular paths, several points along the turn are calculated, and used as a basis for heading determination. By providing the aircraft with regular changes in the desired heading, a smooth circular path is achieved. The number of points in the path is directly proportional to the difference between the desired radius and the minimum radius – if the desired radius coincides with the minimum radius at the current aircraft speed, there is no need to use more than two alternating points to update the heading control. As with the PID approach, this approach also requires frequent communication with the aircraft, in order to send the desired values for the autopilot system.

2.3 AI Autopilot

The third approach makes use of the AI autopilot within the simulator to control the aircraft. This AI autopilot can be used in every aircraft, independently of the existence of an autopilot system such as the one described in the previous approach. It is used by the simulator to guide the generated air traffic from departure to arrival airports.

As with the previous approach, this autopilot also features limitations, namely the bank angle. When an autopilot system is available, the maximum bank angle is used to calculate the minimum radius for a turn, just as in the previous approach. When no autopilot system is present, the default value of twenty-five degrees is used – this is the most common value for autopilot systems, and experimental activities have demonstrated that this is also the most usual value for this AI autopilot.

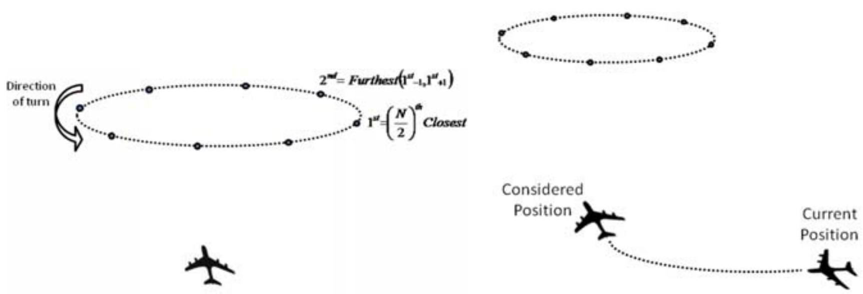
This approach has, however, a few more limitations. In this case, all navigation is based on waypoints, which specify not only the latitude/longitude/altitude coordinates of the desired point, but also the desired speed or throttle percentage to be applied. Having this limitation in mind, and similarly to the previous approach, several points have to be calculated and fed to the aircraft for circular paths. This approach, however, does not require frequent communication with the aircraft – all points of a maneuvering sequence can be sent at the same time, in a waypoint list structure. The exception to this behavior is the existence of loops in the path – if a portion of a path is to be repeated until some external event occurs (such as human intervention), the waypoints that represent that loop have to be sent at the beginning of that loop, with a flag indicating that after the last waypoint, it should return to the first; when the loop is to be broken, the remaining waypoint must then be sent, replacing the previous ones.

2.4 High-Level Maneuvering and Waypoint Computation

In order to compare these methodologies, some high-level maneuvers were considered, namely the 'go to point' instruction, the circle and the helix maneuvers. Additional instructions could be issued to the aircraft, including desired airspeed, heading or altitude and target vehicle interception. These additional instructions would force the aircraft to accelerate or decelerate to attain the desired airspeed, turn and maintain the desired heading or climb or descend in order to reach and maintain the given altitude. The target interception instruction provides the aircraft with the most probable point of interception with the target vehicle, if possible (aircraft and target vehicle speed, distance and heading are used to determine if interception is possible), updating the interception point every few seconds. The authors believe the three first instructions to be sufficient to test the efficiency of vehicle control, since others can be achieved by using the a combination of the first ones or a similar method.

The 'go to point' instruction has three logical parameters: latitude, longitude and altitude of the point to be reached. The altitude is perceived as altitude above the mean sea level, and not altitude above ground level. The circle maneuver, also called loiter, is defined by the central point (latitude, longitude and altitude), the radius of the circle and the number of laps, if not to loop indefinitely. The helix is defined by a central line (latitude and longitude), the initial and final altitudes, the radius of the helix and the number of laps. As previously mentioned, the radius for the last two maneuvers is conditioned by the aircraft's maximum bank angle and airspeed. Also, the number of laps to be executed in the helix maneuver is constrained by the difference between initial and final altitudes and the aircraft's maximum safe rate of climb/descend (vertical speed).

In order to compute the points used by the latter two maneuvers in the two approaches that use autopilot systems, some considerations were made. First, the number of points to consider is directly proportional to the difference between the intended radius and the aircraft minimum turn radius. Second, and most importantly, the order in which those points are presented can deeply influence the path of the aircraft. For a smoother transition, a tangential approach to the virtual circle is preferred, especially when the circle has a radius closer to the minimum turn radius. The following algorithm was used to determine the first point and the order of the remaining points: a list of points is determined for the given radius, ordered as to form a circle; the distances between the aircraft and the points are determined; the first point is the $(N/2)^{th}$ closest point to the aircraft, N being the number of points in the circle; the second point is the point further away from the aircraft, chosen among the two points that are adjoining the first point in the original list of points. The remaining points are determined according to the order in which they were initially determined. Figure 1(a) illustrates the choice of the first and second points in a hypothetical 8-point circle and subsequent direction of turn.


Direction of turn
2^{nd}= Furthest(1^{st}_{-1},1^{st}_{+1})
$1^{st}=\left(\frac{N}{2}\right)^{th}$ Closest
Considered Position
Current Position

(a) Waypoint Determination (b) Aircraft Position Adjustment

Fig. 1. Waypoint Determination and Aircraft Position Adjustment

The current aircraft position is used if the circle or helix maneuver is to be executed immediately; if the maneuver is to be queued after other maneuvers, the final point of the previous maneuver is used. Also, the aircraft position to consider may be adjusted to a position where the aircraft's heading is towards the region of the desired maneuver, considering a turn with the minimum radius at current speed, as illustrated in Fig. 1(b).

3 Experimental Settings

In this section, the experimental settings for the conducted experiments are presented. As already mentioned, FSX was the chosen simulator. Different experimental conditions were recreated for each control approach, with three main controllable variables – aircraft type, weather conditions and maneuver sequence.

Different aircraft types were used to assess how generic the approach was, or if it had limitations, namely regarding the size or type of aircraft. A total of three distinct aircraft were used in the experiments, which include a Piper J-3 Cub (a single-engine two-seater light aircraft, with a wingspan of 10.6 meters and a maximum speed of 74 knots), a Beechcraft Baron 58 (a twin-engine six-seater with a wingspan of 11.5 meters and a cruise speed of 200 knots) and a Bombardier Learjet 45 (a twin-engine-jet nine-seater with a wingspan of 14.6 meters and a cruise speed of 464 knots). Since this project is to be used with small to medium-sized aircraft, the authors felt no need to conduct the experiments with larger aircraft. Although the Piper J-3 Cub does not possess an autopilot system, the authors think it was important to include this aircraft, as to test the performance of the two other approaches on a smaller, slower plane.

Different weather conditions were used to test the control performance under adverse conditions, and to see how flexible the approach was to the existence of uncontrollable and unpredictable external factors. The experiments were conducted under two different weather conditions – fair weather and gray and rainy. The authors felt that harsher weather conditions would not be necessary, since the intended aircraft would not fly under more adverse weather conditions.

Different high-level maneuvering sequences were also tested, as to assess how the approach would handle maneuver transition and how smooth the final flight would be. Basic high-level components were used to compose the three maneuvering scenarios used in the experiments. These high-level instructions include a 'go to point' command, indicating that the aircraft should pass through a given latitude/longitude/altitude point; circle, indicating that the aircraft should circle a given point with a given radius, either one time or continuously; helix motion, either climbing or descending from one initial altitude to a final altitude though a series of turns around a central point.

All experiments begin with the selected aircraft stopped in the end of a runway (34R) of the selected airport – Seattle-Tacoma International – with idled engines, facing the runway. The first command of the maneuvering sequence is always a 'go to point' command, with a point directly in front of the aircraft, approximately 500 feet above the runway, in order to assure both a smooth takeoff and that the aircraft attains its cruise speed. Slight variations in the scenarios were introduced for the three aircraft, due to their different cruising speeds. As already mentioned in the section above, the radius of a banked turn is proportional to the square of the velocity of the aircraft, and hence, a larger radius was used for faster aircraft in the circle and helix commands. As a result, other commands were also modified in terms of longitude/latitude, in order to accommodate for the larger radius of these commands, in an attempt to maintain the overall proportions between the several control points.

Data from the simulation sessions was collected, as to be further analyzed, as described in the following section.

4 Results

In this section, the results that were attained through the experiments that were conducted, and as described in the previous section, are presented. For a more structured arrangement, the results are divided into two categories – communication necessities and maneuver effectiveness.

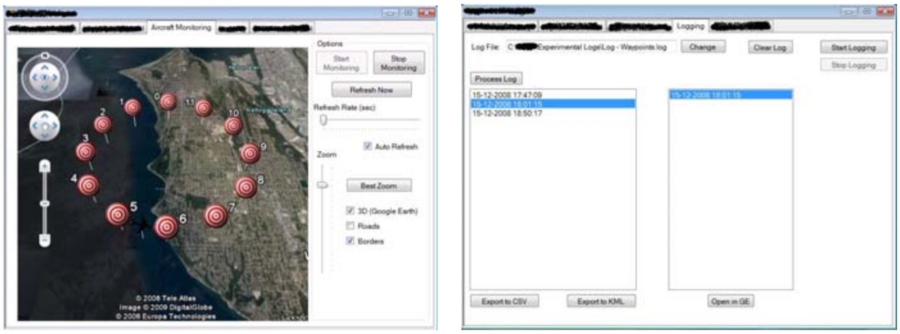
4.1 Communication Necessities

The first dimension that was analyzed is communication requirements. Although this is not a major issue when working in a simulated environment, the future use of real vehicles connected to the platform requires this to be analyzed. In this subject, the third approach is far less demanding. While both the first and second approaches need to communicate with the aircraft at regular intervals, the third approach only communicates in the beginning of the experiment and, in the case of the third scenario, two other times, due to the existence of a user-controlled loop.

In a more detailed view, the first approach needs to send elevator, throttle and aileron data at regular intervals. In the experiments that were conducted, the values were sent every second. Although this was more than enough for the simulated experiments, it is believed that this kind of approach would require, in real live, more frequent adjustments to the aircraft controls to produce a stable flight. The second approach also sent data at regular intervals, even though a higher two-second interval was used. The data sent in this approach consists of heading, speed and altitude values for the autopilot knobs. With the third approach, and considering the mentioned exception of user-controlled loops, all data is sent before-hand. This data consists of a list of a waypoint-describing structure, containing the latitude, longitude and altitude of the point, desired speed or throttle and some flags, indicating how the waypoint should be interpreted.

4.2 Maneuver Effectiveness

Maneuver effectiveness was evaluated on a more subjective level, by analyzing and comparing the paths the aircrafts flew through. On a more immediate level,



Log File: C:[redacted]Experimental Logs\Log - Waypoints.log

```
15-12-2008 17:47:09
15-12-2008 18:01:15
15-12-2008 18:50:17
```

```
15-12-2008 18:01:15
```

(a) Monitoring Tool (b) Logging Tool

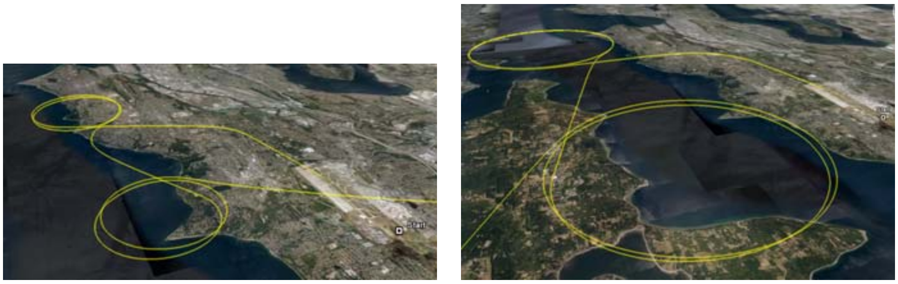
Fig. 2. Monitoring and Logging Tools

the experiment flights were accompanied in real-time, using both the simulator's graphical interface and the developed monitoring tool. This tool uses collected aircraft data as well as information about the determined waypoints to display the current position and orientation of the aircraft, as well as the positions and order of the various waypoints. This was done using the Google Maps and Google Earth APIs, to render the desired icons on top of the two- or three-dimensional representation of the surrounding environment in a plugin using an embedded web browser. Figure 2(a) shows the developed monitoring application, with the several waypoints that define a circle, and a visible aircraft between points five and six. In addition to this immediate and inaccurate visual inspection of aircraft data, information about aircraft position, speed and attitude was collected at regular intervals and stored in a log file. The developed logging application then converted this information into two formats: KML format (Keyhole Markup Language), and CSV format, as can be seen from Fig. 2(b).

The first format, KML, allows for a visual inspection of the paths in an application such as Google Earth. Figures 3(a) and 3(b) show the results of two experiments conducted for the first scenario, using two distinct aircraft.

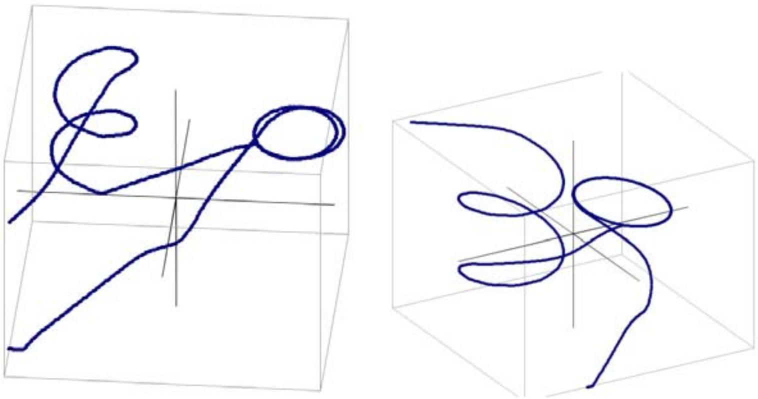
It is clear by analyzing both Fig. 3(a) and 3(b) how a faster aircraft required a change in the location of the circle and helix centers, in order to accommodate the increase of the minimum turn radius, maintaining at the same time a similar overall proportion. The CSV format can be imported to an external application, such as Microsoft Excel, which was then used to generate three-dimensional graphs, as to facilitate in a further, more thorough, inspection of the flight paths without additional visual distractions.

Figures 4(a) and 4(b) show the results from the two experiments above in three-dimensional graphs. A more detailed assessment of the efficiency of the control approaches can be made with these graphs, especially in the altitude dimension. As previously stated, experiments were conducted with three distinct aircraft, using three distinctive scenarios and under two different weather conditions. As to assess the influence of each of these three factors in the



(a) Flight Path A (b) Flight Path B

Fig. 3. Google Earth Preview of Flight Paths A and B



(a) 3D Graph of Flight Path A (b) 3D Graph of Flight Path B

Fig. 4. 3D Graphs of Flight Paths A and B

tested approaches, the results were evaluated by grouping experiments with two common variables and analyzing the results in respect to the third one.

Considering different aircraft, and as previously stated, the second approach could not be tested in the smallest aircraft, since there is no autopilot system available to the pilot. The first aircraft had a similar performance with both the first and third approaches. The second aircraft showed a slightly better performance when using the first and third approaches, compared to the second one. The third and largest aircraft had a comparable performance for all three approaches.

Regarding the three distinct scenarios, there were no significant differences among the three approaches. The three scenarios diverged in the complexity of the intended course, each scenario adding more maneuvers and decreasing the space between maneuvering areas. This caused the aircrafts to make more abrupt turns, but all approaches handled this without posing any problem.

In respect to the influence of weather conditions in maneuver effectiveness, they were similar with all three approaches. The resulting paths were a bit more unstable, presenting more variations in altitude and in some cases widening the radius of the circling maneuvers. In the cases when this happened, the second and third approaches were more susceptible to erroneous maneuvering. In three cases, when using the third scenario, two with the second approach and one with the third approach, the aircraft performed a full circle in order to pass through a point it had previously missed. As for the first approach, the influence of deteriorating weather conditions was also felt in the form of course shifting. This was particularly visible with smaller aircraft, performing circle maneuvers, each lap slightly warped in the direction of the wind.

5 Conclusions and Future Work

In this section, some conclusions that can be withdrawn from the three tested approaches and the experimental results are presented. Also, some lines of future work are presented.

From a more theoretical point of view, one can draw some conclusions from the three approaches that were devised. The PID controller approach is a general method, which works with any aircraft that has the necessary controls (throttle, aileron and elevator). However, it requires an almost constant communication with the aircraft, in order to send the current values. Moreover, most aircraft now have some sort of autopilot system, which performs the same calculations as the PID controller, and, most probably, in a more effective manner, having the gains already tuned for the specific aircraft. The autopilot approach is not a general method, as can be seen from the chosen aircraft for the experimental activities – not all aircraft have autopilot systems. Moreover, it also requires an almost constant communication with the aircraft, just as the first approach. It does, however, require a lot less calculations to be made in the control agent side, leaving them to the autopilot system. The third approach is also a general approach, given that any aircraft can be used with the AI autopilot system. It is far less demanding in what concerns to communications requirements, since it only needs to communicate once. This is, however, a burst in communication, with far more data to be transmitted than any of the other methods. In addition to that, some substantial waypoint calculation has to be previously performed by the control agent.

From a more practical point of view, one has to consider the results that were attained through experimentation, as presented in section 4. In respect to maneuver execution effectiveness, there are no significant differences among the three approaches. However, and although the second approach is viable when dealing with real aircraft, as already explained in section 2.2, it is not practical to use that approach in the simulated environment, since it would exclude some aircraft from being used. The use of different aircraft does not seem to influence the performance of the approaches, with the only visible impact being the increase of the turn radius caused by the increase of the aircraft speed. Also, all approaches seem to be equivalently affected by deteriorating weather conditions, although the autopilot-based approaches are more susceptible to missing one waypoint. This can, however, be easily corrected if one increases the distance at which the aircraft is considered to have reached a waypoint when the weather conditions deteriorate. Regarding communication requirements, one has to conclude that the third approach, based on the AI autopilot, is the best approach, since it does not require a constant communication with the aircraft, and therefore the possibility of error due to a failure in the communication with the aircraft is reduced to a minimum. The first two approaches, which had to communicate every second or once every two seconds, are more susceptible to errors caused by a communications malfunction, which could lead to an erratic maneuver performance, or even to more serious and unpredictable consequences, if the communication breakdown extents for a longer period of time. With the third approach, and even in the case of a complete communication crash, the aircraft would simply continue with the original flight plan and return to the base as originally intended to.

In spite of the project's overall success, some improvement opportunities have been identified. One possible improvement in maneuver definition and execution is the possibility to specify whether the circle and helix movements should be executed banking to the right or to the left. Currently, this is determined by the algorithm as described in section 2.4, resulting in some unpredictability. Another possible improvement would be to automate landing procedures, by automatically selecting the best runway for landing and determining a series of waypoints as to align the aircraft with the runway. Some additional factors that would increase the level of realism of the simulation could be achieved by fault injection, such as the introduction of noise in communication, as well as failures in some aircraft systems, sensors and actuators.

The next step of this project is to select an agent platform as to facilitate communication between agents, to assist in the execution of cooperative missions. When completed, the system will allow for the cooperative planning and execution of missions, such as surveillance of given areas, such as forests (to facilitate early fire detection) or coast lines (to detect and help prevent illegal activities) or even areas affected by pollution, to detect the source and probable progression of the polluting chemicals. The system can also be used to assist in military operations, such as search & rescue operations, covering a larger search area, or enemy surveillance operations.

As a final summary, one can say that the simulation environment shows promising capabilities, namely when referring to structured experiences; the aircraft control approaches were tested successfully, and the future of the project is promising, with many possible useful applications.

Acknowledgment

The authors would like to thank LIACC for providing with all the necessary equipment and for the excellent working conditions.

This work was supported by Project ACORD - Adaptative Coordination of Robotic Teams, FCT/PTDC/EIA/70695/2006. The first author is supported by the *Fundação para a Ciência e a Tecnologia* (FCT), under Doctoral Grant SFRH/BD/36610/2007.

References

1. Stone, R.: Serious Gaming. Defense Management Journal (31), 142–144 (December 2005)
2. Danek, G.L.: Vertical Motion Simulator Familiarization Guide. Technical Memorandum 103923. NASA Ames Research Center, Moffett Field, California, USA (May 1993)
3. National Aeronauticas and Space Administration: Vertical Motion Simulator. NASA Ames Aviation Systems Division (December 2008), `http://www.aviationsystemsdivision.arc.nasa.gov/facilities/vms/index.shtml` (Consulted January 2009)
4. Tran, D., Hernandez, E.: Use of the Vertical Motion Simulator in Support of the American Airlines Flight 587 Accident Investigation. American Institute of Aeronautics and Astronautics Modeling and Simulation Technologies Conference and Exhibit (Aug), Providence, Rhode Island, USA (2004)
5. SimTrain, LLC: SimTrain (2009), `http://www.simtrain.net/index.html` (Consulted January 2009)
6. Samek, J., Sislak, D., Volf, P., Pechoucek, M.: Multi-party Collision Avoidance Among Unmanned Aerial Vehicles. In: IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2007), California, November 2-5, pp. 403–406 (2007)
7. Zhang, X., Xu, R., Kwan, C., Haynes, L., Yang, Y., Polycarpou, M.M.: Fault Tolerant Formation Flight Control of UAVs. International Journal of Vehicle Autonomous Systems 2(3-4), 217–235 (2005)
8. Schurr, N., Marecki, J., Tambe, M., Scerri, P., Kasinadhuni, N., Lewis, J.P.: The Future of Disaster Response: Humans Working with Multiagent Teams using DEFACTO. In: American Association for Artificial Intelligence (AAAI). Spring Symposium on AI Technologies for Homeland Security (2005)
9. Gimenes, R., Silva, D.C., Reis, L.P., Oliveira, E.: Flight Simulation Environments Applied to Agent-Based Autonomous UAVs. In: Proceedings of the Tenth International Conference on Enterprise Information Systems, SAIC, pp. 243–246 (2008)
10. Stark, P.: FSX Mission Building, Parts 1-4 PC Pilot Magazine. Key Publishing Ltd. (48), 40–43 (September); (49), 40–44 (November); (51), 40–44 (January 2008); (52), 40–44 (March 2007/2008)
11. Microsoft Corporation: Microsoft ESP - A New Era in Visual Simulation (2008), `http://www.microsoft.com/esp/` (Consulted January 2009)
12. Training & Simulation Journal: Lockheed Martin, FlightSafety to use Microsoft ESP platform. Training & Simulation Journal Online (February 21, 2008), `http://www.tsjonline.com/story.php?F=3384514` (Consulted January 2009)
13. Chao, H., Cao, Y., Chen, Y.: Autopilots for Small Fixed-Wing Unmanned Air Vehicles: A Survey. In: International Conference on Mechatronics and Automation 2007 (ICMA 2007), pp. 3144–3149 (2007)

14. Erdos, D., Watkins, S.E.: UAV Autopilot Integration and Testing. In: IEEE Region 5 Conference, April 2008, pp. 94–99 (2008)
15. Skogestad, S.: Simple Analytic Rules for Model Reduction and PID Controller Tuning. Journal of Process Control 13(4), 291–309 (2003)
16. CloudCap Technology: A Brief History of Piccolo Development CloudCap Technology, Hood River, Oregon, USA (June 2008)

Adaptive Serious Games Using Agent Organizations

Joost Westra, Hado van Hasselt, Frank Dignum, and Virginia Dignum

Universiteit Utrecht

Abstract. Increasing complexity in serious games and the need to reuse and adapt games to different purposes and different user needs, requires distributed development approaches. The use of software agents has been advocated as a means to deal with the complexity of serious games. Current approaches to dynamic adjustability in games make it possible for different elements to adjust to the player. However, these approaches most use centralized control, which becomes impractical if the complexity and the number of adaptable elements increase. The serious games we are investigating are constructed using complex and independent subtasks that influence each other. In this paper, we propose a model for game adaptation that is guided by three main concerns: the trainee, the game objectives and the agents. In particular we focus on how the adaptation engine determines tasks to be adapted and how agents respond to such requests and modify their plans accordingly.

1 Introduction

Increasing complexity of software games, in particular of serious games [12, 13], and the need to reuse and adapt games to different purposes and different user needs, requires distributed development approaches. As games become more and more sophisticated in terms of graphical and technological capabilities, higher demands are also put on their content and believability. The use of software agents has been advocated as a means to deal with the complexity of serious games [8]. In these often many different characters performing complicated tasks interact with each others and with the trainee. In the case of the fire command training game, one could think of a situation where characters playing the role of fire agents need to be replaced by soldiers (for instance, when the disaster increases to a very complex situation). It would be very useful if this could be done without rewriting the complete strategy. Serious games are applications developed with game technology and game design principles for non-entertainment purposes, including games used for educational, persuasive, political, or health purposes. The goal of a serious game is to teach certain pre-specified tasks to the trainee of the game (the trainee). In serious games, quality is measured in terms of how well the components in the game are composed, how they encourage the player (or trainee) to take certain actions, the extent to which they motivate the player, i.e. the level of immersiveness the game provides, and how well does the gaming experience contributes to the learning goals of the trainee [3].

Believability is a main driver of game development. The search for enhanced believability has increasingly led game developers to exploit agent technology in games [8]. In particular, dynamic response, multiple conflicting goals, team work and cognitive models are all agent research issues that are relevant for game developers. Furthermore, serious games need to be suitable for many different people. Currently this adjustment is either done externally by experts that need to guide the adaptation, or the game provides a number of predefined levels based on (learner) stereotypes. While the use of expert guidance to adaptation usually results in quite effective training, such experts are rare, not available during the training session, and expensive. The use of predefined adaptation levels may lead to less optimal adaptation in the case trainees do not fit well with the stereotypes. Furthermore, the enhancement of the learning experiences requires to (automatically) adjust the game online.

Three requirements for online game adaptation have been identified [1]. First, the initial level of the player must be identified. Second, the possible evolutions and regressions in the player's performance must be tracked as closely and as fast as possible. Third, the behavior of the game must remain believable. In this paper we will mainly concentrate on the third aspect while trying to achieve the first two. In order to optimize learning, serious games should provide the trainee an ordered sequence of significantly different and believable tasks. Without a clear organization structure, adaptation can quickly lead to a disturbed storyline and the believability of the game will be diminished. Furthermore, characters in serious games are usually active for relatively long periods in serious games. This poses an extra burden on the believability of the game, namely coherence of long-term behavior [9].

The realization of the tasks of the trainee require the coordination of the actions of many different characters. For example, a serious game for training a fire commander includes scenarios aiming at learning how to make sure the victims in a burning building are saved. Adaptability to learning objectives implies these characters to show a spectrum of behavior. The victims could be more or less mobile or they could be located in simple or difficult locations. There may be bystanders that could obstruct the medics. The police could autonomously control the bystanders or could only act if ordered by the fire commander. Also the behavior of the medical personnel affects the difficulty of the task. As becomes clear from this example the difficulty of the trainee's task is very much dependent on the behavior of all the characters in the scenario, resulting in many variations of the same global task.

Performance of each subtask can not be measured separately because all behaviors influence each other. If all characters are allowed to adapt to the trainee without coordination, situations will occur where all adapt simultaneously resulting in unwanted scenarios for the trainee. For example, all victims become less mobile, all police agents become less autonomous and bystanders provide higher obstruction to medics. Consequently, adaptation should be coordinated according to the learning objectives while maintaining a coherent storyline. In previous work [15] we proposed the use of multi-agent organizations to define a storyline in such a way that there is room for adaptation while making sure that believability of the game is preserved. In this paper, we discuss the effect of these approach to adaptation on the design of the agents.

The paper is organized as follows. In the next section, we introduce GAM, the model for game adaptation. The background and motivation for this model are discussed in section 3. In section 4 we describe the task selection mechanism based on user and game model, and in section 5 the consequences of the adaptation model in terms of agent architecture are described. Extensions and conclusions are discussed in the last section.

2 Game Adaptation Model

Systems of learning agents are usually very unpredictable. This is not a problem for applications like simulations or optimizations where only the end result matters. But for games, where the agents are adapting to the trainee during the game and thus the adaptation must have a direct beneficial influence, the system needs to be a lot more predictable. Therefore, adapting the game to the trainee in complex learning applications requires both learning capabilities and decentralized control. In order to guarantee the successful flow of the game and the fulfillment of the learning objectives, the system also needs to be able to describe and maintain global objectives and rules.

Adaptation in games is guided by three main concerns: the trainee, the game objectives and the agents. The trainee, its capabilities, learning objectives and learning style are central to the adaptation. The purpose of the game is to provide a suitable environment for the trainee, so the primary requirement for adaptation is the need to determine trainee's initial state, objectives and style. As important is the game setup. In order to ensure believability and therefore contribute to the learning experience of the trainee, adaptation should maintain coherence of the storyline and integrate seamlessly to the scene where the trainee is situated. In our approach, agents provide the behavior for different game elements. Agents can be behind a game character, but also control some other (non living) elements, such as the strength of the fire and the quality of obstacles in the fire commander training example given above. Each agent pursues its own goals, and should be able to adapt its behavior by providing different plans for a goal or a range of alternative goals. GAM (Game Adaptation Model) includes the elements described above as depicted in figure 1.

Based on the current state of the trainee and on the current moment in the game storyline, the adaptation engine should be able to determine what type of behavior should be requested from the agents. Based on this request, each agent can determine its own possibilities for adaptation (or a range of possibilities) that should fit with the agent's own goals and its own perception of its role in the storyline. Based on the proposals by the agents, the adaptation engine will determine the overall adaptation and request the chosen agents to change their behavior accordingly. Finally, the resulting game scene is presented to the trainee.

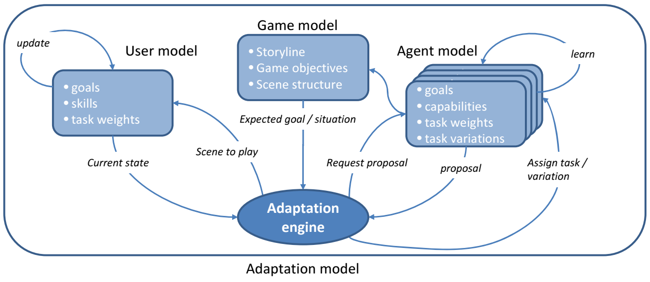


Fig. 1. The Game Adaptation Model

In training applications it usually is possible to make an estimation of the difficulty of a certain task. This can be done using domain expert knowledge that associates different task implementations with a specific difficulty level. These estimates can be updated in an offline learning phase and will get more accurate if more users have used the application. Having this information in advance significantly speeds up the adaptation because the algorithm not only needs to learn the strength relations between the different task implementations but is even able to directly select appropriate implementations for a given trainee skill level. The coordination of the difficulty of the tasks of the different agents is done using the organizational description of the storyline as given in the Game Model. If the trainee is performing above expectation and all the agents decide to increase the difficulty at the same time the application will probably be too difficult the next time. The adaptation engine will determine how much agents are allowed to adapt.

3 Background

As described in the previous section, we advocate to bring together three issues in order to adapt serious games to the user. The adaptation should be distributed over the separate elements that constitute the story line, these elements should adapt themselves online using some machine learning technique and they should do it in an organized fashion to maintain the general story line. In this section, we discuss the related work that can be used for these aspects.

3.1 Adaptation in Games

Even though many commercial games do not use any adaptation [10], already some research has been done on adaptation in games. However, most of this research focuses on adaptation of certain simple quantitative elements in the game. For example better aiming by opponents or adding more or a stronger type of opponents.

Most commercial computer games that have varying degrees of difficulty do not use online adaptation. They have a number of preset difficulty levels that need to be selected in the beginning of the game. Offline learning could be used for these predefined levels but usually these are just scripted by hand. Current research on online adaptation in games is based on a centralized approach [6, 14]. Centralized approaches define the difficulty of all the subtasks from the top down. This is only feasible if the number of adaptable elements is small enough and if the separate adaptable elements have no separate time lines that need to be taken into account. In shooting games, for example, these requirements are not problematic. The games only adapt to the shooting skill of the trainee and most characters only exist for a very limit period of time.

Another important aspect of adaptation in (serious) games is the distinction between direct and indirect adaptation. Direct adaptation occurs when the designer specifies possible behavior of the agents in advance and specifies how and when to change this behavior. The designer also specifies what input information should be used. Direct adaptation only allows adaptation to aspects that the designer has foreseen. No unexpected behavior can emerge when using direct adaptation. On the other hand, in indirect adaptation performance is optimized by an algorithm that uses feedback from the game world. This requires a fitness function and usually takes many trials to optimize. If indirect optimization is used the algorithm also needs to be able to cope with the inherent randomness of most computer games. In this paper, we will use an approach that has the benefits of direct adaptation without the need for the designer to directly specify how the adaptation should be done. The designer is able to specify certain conditions on the adaptation to guarantee the game flow but does not have to specify which implementations are chosen after each state.

Research has been done on using reinforcement learning in combination with adaptation to the user [2, 14]. Most of these algorithms rely on learning relatively simple (sub-)tasks. Moreover, the aim of these adaptation approaches is learning the optimal policy (i.e. making it as difficult as possible for the user). In order to avoid that the system becomes too good for the user, some approaches filter out the best actions to adjust the level of difficulty to the user. This results in unrealistic behavior where characters that are too successful suddenly start behaving worse again. Little attention is paid to preserving the story line in present online adaptation mechanisms, because they only adjust simple (sub-) tasks that do not influence the storyline of the game. Typical adjustments are for example, changing the aiming accuracy of the opponents or adding more enemies. However, even when adding more enemies the algorithm should already take into account that they can only be added in unexplored areas and do not influence the progress of the game.

3.2 Agent Organizations

Adapting the game to the trainee for complex learning applications requires both learning capabilities and decentralized control. However, in order to guarantee successful flow of the game and the fulfillment of the learning objectives, the system needs to be able to describe global objectives and rules. Although many applications with learning agents exist, multi-agent systems with learning agents are usually very unpredictable. This is not a problem for applications like simulations or optimizations where only the end result matters. But for (serious) games, where the agents are adapting during the game and thus the adaptation directly has to have a beneficial influence, the system needs to be a lot more predictable. Systems with this type of characteristics have been successfully modelled using agent organization frameworks such as OperA [5]. In this framework it is possible to define conditions when certain plans are allowed or not. The ordering of the different possible plans can also be defined in this framework. This allows the designer to make sure that the users are not exposed to tasks that are not suitable yet or would ruin the storyline. In previous work we have shown how to use agent organizations to specify the boundaries of the game [15].

The OperA model for agent organizations enables the specification of organizational requirements and objectives, and at the same time allows participants to have the freedom to act according to their own capabilities and demands. In OperA, the designer is able to specify the flow of the game by using landmarks. The different sub-storyline definitions of the game are represented by scenes (the boxes in the figures are separate scenes) which are partially ordered without the need to explicitly fix the duration and real time ordering of all activities. That is, OperA enables different scenes of the game to progress in parallel. In the scenes, the results of the interaction are specified and how and in what order the different agents should interact.

Such an interaction structure defines the ordering of the scenes and when it is allowed to transition to the next scene. The scenes are defined by scene scripts that specify which roles participate and how they interact with each other. The definition of the organization can be so strict that it almost completely defines the strategy. But it is also possible to specify the organization in such a way that all the agents in the game work towards achieving the goals of the game but are still able to do this using different strategies. It is also possible to define norms in the scene description. This makes it possible to put extra restrictions on the behavior of the agent. In a scene script, it is also possible to define certain time constraints to make sure that the game progresses fast enough.

3.3 Adaptation with BDI-Agents

Autonomous game characters should be able to ensure that they maintain a believable storyline. They also have to make complex decisions during to game because not all information is known before the game started. Using BDI agents is a suitable implementation because it allows us to create intelligent characters that are goal directed and able to deliberate on their actions. For the implementation of the BDI agents we will use the 2apl [4] language. 2APL is an effective integration of programming constructs that support the implementation of declarative concepts such as belief and goals with imperative style programming such as events and plans. Like most BDI-based programming languages, different types of actions such as belief and goal update actions, test actions, external actions, and communication actions are distinguished. These actions are composed by conditional choice operator, iteration operator, and sequence operator. The composed actions constitute the plans of the agents. Like existing agent programming languages, 2APL provides rules to indicate that a certain goal can be achieved by a certain pre-compiled plan. Agents may select and apply such rules to generate plans to achieve their goals.

Most BDI architectures do not provide learning abilities. However, an extension of 2apl for learning agents is available, in which multiple equivalent 2apl plans are available that are suitable for different skill levels [7].

4 Game Adaptation Engine

As discussed in the previous sections, current work on game adaptation does not fulfill the requirements of complex distributed serious game applications. We propose an adaptation approach that uses expert knowledge for the adaptation but is more flexible than current direct adaptation methods.

Important to keep in mind is that the task that needs to be executed by the trainee follows from the behavior of the agents. Each agent is responsible for adapting its own subtask(s) while making sure that believability is preserved. Think of the example where the trainee is a fire commander that needs to secure victims inside a burning building. Example of a subtask is controlling the fire, which can easily be increased in difficulty by letting the fire spread and grow more or less rapidly. However it would be unrealistic if the fire suddenly doubles in size without a gradual increase. In the same way, many more agents' subtasks influence the performance of the trainee. The bystanders could be obstructing the medical personal more if time progresses but it would be unbelievable if the number of bystander suddenly doubles. A typical example of subtasks directly interacting with each other occurs when there is a police agent that has the ability to influence the bystanders. These are however subtasks from the training perspective. The trainee needs to give orders to the police man if he is not performing well while dealing with the bystanders is a different task.

4.1 Task Selection

The goal of the game engine as depicted in figure 1 is to select the best tasks that suit the needs of the user. Speed of adaptation is important in games to ensure a continuous flow, therefore, to make adaptation as fast as possible we use a flexible form of direct adaptation in the online phase. This means that all the different possible behavior variations for all agents are implemented a priori and stay fixed. The adaptation task is then to select a subset from this set of possible behaviors that is most suitable task for the user at a given moment. It should be noted that the tasks to be performed by the trainee are dependent on the game environment but even more on the possibilities provided by the (adaptive) agents. These agents can perform different plans based on task preference. For example, a task could be made easier if a cooperative agent is autonomously performing a large portion of the task without requiring lots of input from the user.

Assuming an user model indicating the user aims, capabilities and learning objectives, the game engine will select a task that is most suitable for the trainee but also preserves the storyline and gameflow. We assume that the storyline is defined by an agent organization framework, such as OperA. At the beginning of the game, a default user model is used, which may be not a very accurate representation of the trainee but this model will be adapted according to the trainee's performance. On the other hand, each agent has it own preferences and these preferences are likely to conflict with each other. We have chosen to use a kind of combinatorial auction [11] to select the best suitable plans for all the agents.

The main objective of the adaptation engine is to choose the most optimal task for the trainee, given the user model, the game flow and the capabilities of the agents. The task of the trainee is created by combining the subtasks of the agents. The agents representing the different adaptable elements will make different possible variants of their subtasks available that can be executed in the next time step. The actual selection is thus dependent on the required task for the trainee and on the preferences of the agents.

Figure 1 shows a schematic overview of the whole task selection process. This is optimized on the skills of the trainee but also most fitting with the preferences of the agents. The preferences of the agents are different because they have as a goal to stay as consistent and believable as possible and to follow their own preferred ordering of tasks. The agent uses information from the Game Model where the ordering of tasks can be specified. During this task selection we also use the restrictions from the Game Model to select a fitting task according to the organizational requirements. An example of this is that in the organization it is specified that one agent always goes left and one agent goes right. In the case that one agent specified no preference while the other agent prefers to go right, the agent without a preference should select the variations where it goes left.

The resulting "selected task" is a combination of all the plans that will be executed by the agents. This task has a certain combined difficulty for all the separate learning tasks. After the task is completed this information can be used to update the user model. The update of the user model is also dependent on the performance of the user.

4.2 Formalization

In this section we will formalize some important aspects of a training simulation. Of each of these aspects, we will discuss how to obtain the values of these.

Definition 1. ***(Training)***
The complete training consist of a partially ordered set of tasks:

(D1) $M = (\{t_T^i\}, P_c)$

with P_c the partial ordering.

For a given task T,

(D2) $V_T = \{t_T^1, ..., t_T^n\}$

are the possible variations of T*.*

Each variation t_T^i *is formed as a partially ordered set of subtasks:*

(D3) $t_T^i = (X_T^i \subseteq S, P_T^i)$

The partial ordering P_x *is selected from a number of predefined partial orderings* $V_P = \{P_1, \ldots, P_n\}$ *belonging to each task* T*.*

S *consists of all possible atomic subtasks:*

(D4) $S = \{s_1, \ldots, s_x\}$

Where each subtask corresponds to an optional agent plan.

Subtask belong to different skill categories that need to be learned by the trainee. Skills correspond to a certain user task type such as *extinguishing a fire* or *giving team orders.*

Definition 2. ***(Difficulty)***
The difficulty for a variation is determined separately for each skill l*. The difficulty for a skill is only dependent on the subtasks that correspond to that skill:*

(D5) $d(t_T^i, l) = g(\{hd(s) | s \in X_T^i \wedge L(s) = l\})$

Function g determines the total difficulty for a skill category from the difficulty of the separate subtasks that belongs to the corresponding skill category. Currently function g uses the average difficulty but this can be improved in the future. Function hd determines the difficulty of the separate tasks, this depends on the selected subtask execution and are defined by a human expert. These values vary between 0 and 1. These values indicate the expected needed skill of a trainee to perfectly complete the task. For instance, on a task with a value of 0.5 on some property, a trainee with a skill level of 0.2 is expected to have difficulty completing the task, while a trainee with a skill of 0.6 should be able to perform the task perfectly. The assumption is that tasks with a skill level a little higher than the skill level of a user for certain properties can be expected to be challenging enough to be valuable for training, but not too hard to complete.

Definition 3. ***(Weights)***
The weights for each skill category are dependent on the partial ordering P_x*:*

(D6) $wgt(t_T^i, l) = hw(P_T^i, l)$

The possible partial ordering and the corresponding weight for each skill category are predefined by a human expert (function hw). These values indicate how much each skill is needed for the task. These weights will be used to determine how much to update each skill level of a trainee that performed the corresponding task. We require each weights value to be a real valued number

$0 \leq hw(P_T^i, l) \leq 1$. There are two equivalent ways to establish these weight variables in a useful way. In the first case, each weight corresponds to the amount of impact that each property has on the training as a whole. For instance, one way to determine such weights is to use the expected amount of time needed for a certain subtask corresponding to a certain skill. Note that this is an orthogonal concept to difficulty, because subtasks that make up the largest part of a task are not necessarily also the hardest parts of that task. In the second case, a perhaps easier way to handle these weights variables is to separate the impact on learning as a whole from the make up of a certain task. Then, the weights of a task could be required to sum to one over all skills. The interpretation of a weight then is the part of that skill that is needed to perform this task. Such a tuple may be somewhat easier to construct, for instance by human experts. However, an indication of the size of the task compared to the training as a whole is needed, that should be stored in a different number V.

A reinforcement function is needed to evaluate the performance of the trainee. Such a function should be defined for each task, such that it can give feedback about the level of performance of the trainee on that task. The output of a reinforcement function is a number r, that can range from 0 to 1, where the interpretation is that when a trainee receives a reinforcement of $r = 1$ the task was performed perfectly. Conversely, a reinforcement of $r = 0$ indicates that the trainee failed to complete the task at all. We only consider reinforcement functions that are constructed by human experts, although for most domains some metrics such as the time needed to complete a task compared to the average time needed to complete the task by similar users could be used as reinforcement.

Definition 4. ***(User Model)***
This model is a tuple of size l:

(D7) $U = (u_1, u_2, \ldots, u_l)$

where each element indicates the expected skill level of the user on each corresponding skill category.

This model is hard to obtain a priori, since it would require a screening of each user on all the relevant skills. Therefore, in the next section we will discuss ways to update this model automatically.

The game organization model describes which combinations of subtasks are allowed at any state of the game. Each agent is able to provide some of the subtasks at a certain difficulty level. Furthermore, the user model will indicate the desired difficulty level for each skill category for the user task, $d_{U,p}$. Then, the objective of the adaptation engine is to determine a ordered set of subtasks that is allowed according to the game model, possible to be executed by a set of agents, and which difficulty is $\forall l : dfclt(t_T^i, l) = d_{U,l}$.

4.3 Updating User Models

As mentioned in the previous section, a user model is a tuple of size l containing estimates of skill levels for the user. For instance, these could be instantiated with a value of 0 for each skill, meaning we do not expect the user to be able to perform any skill to a desired level yet. For instance, these could be instantiated with a value of 0 for each skill, meaning we do not expect the trainee to be able to perform any skill to a desired level yet. We now allow a trainee to try to perform some scenario. This scenario will consist of a number of tasks that can potentially be divided further into subtasks. For each task we assume the difficulty $D = (d_1, \ldots, d_l)$ (using $d_i = dfclt(t^i_T, i)$) and weights $W = (w_1, \ldots, w_l)$ (using $w_i = wgt(t^i_T, i)$) are calculated and stored in tuples and that a reinforcement function exist. After a trainee completes any subtask for which each of these aspects is defined, we can update the user model as follows:

$$\forall u_i \in U: \quad u_i = (1 - \alpha_i w_i)u_i + \alpha_i w_i r \max(d_i, u_i), \tag{1}$$

where $0 \leq \alpha_i \leq 1$ is a step size learning parameter. First assume that $u_i < d_i$. Since we required that w_i is between 0 and 1, the effect of this update is as follows: if the task was performed perfectly, $r = 1$ and u_i will get updated towards d_i with a step size dependent on α_i and w_i. If, on the other hand, the task failed completely, the update results in an update of u_i towards zero. It can be easily verified that for skills that are not needed for the task the trainee skill is not updated, since w_i is then equal to zero. Typically, we will want α_i to be reasonably high in order to learn quickly from the received reinforcements. It is however possible to set α_i to zero, for instance when the main goal of a certain task is to only update specific parts of the user model.

When $u_i > d_i$, the trainee is assumed to be able to perform a task with difficulty d_i perfectly. If that is indeed the case, $r = 1$ and the trainee is updated towards u_i. This implies the user model is not updated. However, if the trainee is not perfect and $r < 1$, the trainee will get a negative update that is dependent on how low the reinforcement in fact is. This update should ensure that the skill levels in the user model will converge to the actual skill of the user. We note that the expected user skill u_i will get updated positively every time that $rd_i > u_i$. This implies that for difficult tasks, a trainee with low skill does not need to perform perfectly to increase the expected skill. In general, we want users to increase their skill levels, so we want them to perform tasks where the expected reward is higher than u_i/d_i. If we are correct in estimating these expected reinforcements, we can easily determine which tasks are fruitful options for training. This brings us to the following important point: estimating the expected reinforcements.

5 Agent Perspective

5.1 Action Selection

Agents are modelled according to the BDI architecture, which allows for plan and goal selection. In particular, we use a modified version of 2APL that provides learning capabilities to the agents [7]. It should be noted that in our approach, the individual agents are not responsible for the learning rate of the trainee.

That is taken care of by the adaptation engine. However, the goals of the agents are explicitly determined to facilitate the trainee's objectives. Furthermore, we assume that all agents aim at being as helpful as possible and rather act in a scene, to support the trainee, than they are idle. This means that the agent is primarily responsible for fulfilling its own goals. This means that all tasks performed by the agents contribute in some way to the learning of the trainee, but agents are not directly responsible for finding the right combination of plans or to coordinate their tasks with the other agents.

Usually, the agent will propose to the adaptation engine multiple different plans that it would like to execute, according to its goals. We make a distinction between different types of plan sets that the agent can propose. At the highest level we have plans that are different because they are designed to fulfill different goals. Next, we have plans that are in a different class because they are designed to fulfill the same goal but use different action types to achieve this. For example, an agent could have multiple actions for satisfying the goal to extinguish a fire. It can achieve this by using water, or oxygen deplete the fire by using explosives. And finally we have multiple plans that use the same action type but have a different execution of this action that is similar but are created to propose different difficulty levels. For example, there could be multiple implementations of using a water hose to extinguish a fire with different degrees of effectiveness.

Based on the information on the required difficulty of the subtask set, provided by the adaptation engine, an agent will generate one or more fitting proposals. The agent has two conflicting concerns when proposing actions to the adaptation engine. One the one hand, the agent has certain preferences related to keeping its behavior as consistent and believable as possible. On the other end should the agent also propose actions that allow for a suitable combination for the trainee. Suggesting suitable plans is not only dependent on the desired difficulty but also on information of the current environment and of past occurrences. For example, the agent can observe that another agent is already performing a certain task and the agents knows that only one agent can perform this task at the same time. The numbers of plans proposed by the agents have a big influence on the whole system. In the extreme case that each agent only proposes one option, the adaptation engine can only select which agents participate in the scene not what behavior they perform and which variation is used. This allows for little possible adaptation to the user and a higher chance that no valid combinations can be made. An advantage of proposing very few options is that the behavior of the agent is more believable because the agents propose actions that are best fitting with their own preferences (staying as believable as possible). In the other extreme where the agents propose all valid actions, there are a lot more possible combinations, allowing multiple valid solutions and more possibilities to adapt to the user. A disadvantage is that the agents could end up performing actions that do not fit their storyline very well. The agents are able to give preferences on the actions they propose but it depends on the adaptation engine to which degree these preferences are used. The optimal solution will lie somewhere between these extremes. Agents should provide enough plans to allow for proper adaptation to trainee but not propose unrealistic plans, while the adaptation engine should keep the preferences of the agent into account.

The adaptation engine not only selects the best matching combination of plans but also is able to influence plans proposed by the agents to achieve a better combination for the trainee. If the organization detects the trainee would be best served if more agents suggest plans related to a certain action type at a certain time, it can request this from the agents. Remember that the agents already propose all reasonable options so this request should not be used to try to let the agents propose different plans without a special reason. One reason could be that there is no valid combination possible with the current bids, but this should not happen frequently. A situation that will occur more often is that agents are performing actions that continue for longer periods of time but can be terminated upon request. If this is not the case, the organization could have the agent switch plans with an active plan that should not be terminated at that time. Or if the organization always waits for the agents to finish its active plan before assigning a new plan, some plans that very easily could be switched might not be executed because the agent is performing a simple plan that takes a very long time (possible infinite) to complete.

Both the timing of switching plans as well as the decision to comply with a request from the adaptation engine are the responsibility of the agents. In some cases the agent is able to perform certain plans (and thus can be requested by the organization) that at that time would not fit the storyline of the game very well. Even though the expected combined difficulty of the task might be less suitable for the trainee if the agent does not comply to the request, it could still be a wise decision if the storyline of the game is better preserved.

We assume that when the default internal reasoning of the agents suggests preferences on actions, the behavior of the agent is more natural. This provides an advantage when compared with top-down orchestration of all the plans.

Definition 5. ***(Agent Proposal)***
An agent proposal is defined as a set of actions with their corresponding preference:

(D5) $Prop = \{(s, pref(s)) | x_l \leq hd(s) \leq y_l \wedge s \in S_{ok}\}$

With x_l and y_l defining the boundaries of the difficulty and S_{ok} defines the actions that are legal in regards to the storyline.

5.2 Learning Agents

Agents are able to adapt their behavior in order to optimize achieving its own goals. On the level of individual adaptation, it is preferable that the agents do not learn completely new plans, as learning new plans is slow in online adaptation. Another disadvantage of learning new plans is that it requires exploration which might actually degrade performance. This leaves the agent with the ability to learn which of its predesigned behaviors to select. The performance measure for this task is separate from the adaptation model. The agent still needs to explore different actions but all the available actions are viable options.

Three different methods can be used when doing this online adaptation. The most simple approach is to predefine expected outcomes for each action type. If the performance is worse than the predefined expected outcome then the agent should try a different action type. For tasks which can have the same feedback parameter for the different actions, reinforcement learning can be used to optimize the agents performance.[1] Agents can use reasoning for selecting different action types. For example, if the agent knows that a truck is blocking the road than it could better execute a different action where access to the road is not needed.

6 Conclusion

In this paper we discussed online adaptation in serious games. The basis for the adaptation lays with the use of learning agents. In order to coordinate the adaption of the agents we use an organizational framework that specifies the limitations of the adaptation in each context. We have shown how GAM, the Game Adaptation Model, meets the requirements posed on adaptation of the game. I.e. it is done on-line, takes care of a natural flow of the game and optimizes the learning curve of the user. In order to fulfill the requirements the GAM uses a user model, the preferences of the agents and uses the guidelines from the organization model.

We argued that an agent based approach for adapting complex tasks is more practical than a centralized approach. It is much more natural if the different elements are implemented by separate software agents that are responsible for their own believability. For complex games where characters play roles over extended periods of time this increases the believability of the whole game. However, the system does not only need to be flexible, the designer also must be able to define the storyline and put certain restrictions on the combined behavior of the agents. Agent organization frameworks are very suitable for creating a flexible system that still must follow a certain progression and behave according to certain norms. Hence the use of OperA for the specification of the organization of the agents in the game.

The proposed model for game adaptation selects tasks that are most suitable for the trainee while following the specification of the game model and keeping the preferences of the separate agents into account. The combination of adaptations selected at each moment is done through a kind of auction mechanism that provides a balance between local optimization of the task and believability of the agent and overall difficulty of the situation for the trainee.

[1] It is beyond the scope of the article to discuss the implementation of the adaptation using reinforcement learning. As we discussed in the related work section we have already done some experiments with online learning using an extended version of 2apl. The learning will go very quickly because of the limited number possible action type.

The next steps will be the actual testing of a complete serious game on firefighting with actual firefighters in the Netherlands and a professional simulation environment provided by VSTEP using the Quest3D engine. This is set up to take place in the near future.

References

1. Andrade, G., Ramalho, G., Gomes, A.S., Corruble, V.: Dynamic game balancing: An evaluation of user satisfaction. In: Laird, J.E., Schaeffer, J. (eds.) AIIDE, pp. 3–8. The AAAI Press, Menlo Park (2006)
2. Andrade, G., Ramalho, G., Santana, H., Corruble, V.: Extending Reinforcement Learning to Provide Dynamic Game Balancing. Reasoning, Representation, and Learning in Computer Games (2005)
3. Brusk, J., Lager, T., Hjalmarsson, A., Wik, P.: Deal: dialogue management in SCXML for believable game characters. In: Future Play 2007: Proceedings of the 2007 conference on Future Play, pp. 137–144. ACM, New York (2007)
4. Dastani, M.M., Meyer, J.-J.C.: A practical agent programming language. In: Dastani, M.M., El Fallah Seghrouchni, A., Ricci, A., Winikoff, M. (eds.) ProMAS 2007. LNCS (LNAI), vol. 4908, pp. 107–123. Springer, Heidelberg (2008)
5. Dignum, V.: A Model for Organizational Interaction: based on Agents, founded in Logic. SIKS Dissertation, series
6. Hunicke, R., Chapman, V.: AI for Dynamic Difficulty Adjustment in Games. In: Proceedings of the Challenges in Game AI Workshop, Nineteenth National Conference on Artificial Intelligence, AAAI 2004 (2004)
7. Kok, E.: Learning Effective Rule Selection in 2APL Agents. Technical report, Master thesis, Utrecht University
8. Lees, M., Logan, B., Theodoropoulos, G.: Agents, Computer Games and HLA. International Journal of Simulation Systems, Science and Technology
9. Moffat, D.: Personality parameters and programs. In: Petta, P., Trappl, R. (eds.) Creating Personalities for Synthetic Actors. LNCS, vol. 1195, pp. 120–165. Springer, Heidelberg (1997)
10. Rabin, S.: AI Game Programming Wisdom. Charles River Media (2002)
11. Sandholm, T.: Algorithm for optimal winner determination in combinatorial auctions. Artificial Intelligence 135(1-2), 1–54 (2002)
12. Schurr, N., Marecki, J., Lewis, J.P., Tambe, M., Scerri, P.: The DEFACTO system: Training tool for incident commanders. In: Veloso, M.M., Kambhampati, S. (eds.) AAAI, pp. 1555–1562. AAAI Press / The MIT Press (2005)
13. Silverman, B., Bharathy, G., O'Brien, K., Cornwell, J.: Human behavior models for agents in simulators and games: part II: gamebot engineering with PMFserv. Presence: Teleoperators and Virtual Environments 15(2), 163–185 (2006)
14. Spronck, P., Ponsen, M., Sprinkhuizen-Kuyper, I., Postma, E.: Adaptive game AI with dynamic scripting (2006)
15. Westra, J., Dignum, F., Dignum, V.: Modeling agent adaptation in games. In: Proceedings of OAMAS 2008 (2008)

Intelligent Agent Modeling as Serious Game

Towards Integrating Microworlds, Tutoring and Evolution

D.W.F. van Krevelen[⋆]

Section Systems Engineering — Faculty of Technology, Policy and Management
Delft University of Technology, Jaffalaan 5, 2628 BX Delft, The Netherlands
d.w.f.vankrevelen@tudelft.nl{tudelft.nl/dwfvankrevelen}

Abstract. Educators increasingly turn to serious games to let students explore complex worlds in a safe environment. In serious games for ill-defined problem domains such as infrastructures and markets, students often interact with preconceived agents at an operational level. We hypothesize however that students could discover more about a domain's complexity at a strategic level by building and testing their own delegate agents. Testing this requires an environment where students and teachers can construct agents at their own level of expertise with recent modeling technologies. For instance, students may create agents not just directly, by building or modifying comprehensive agent models with visual programming languages, but also indirectly, by shaping agent behavior as it evolves in user-defined training scenarios or by enacting example behavior which agents learn to imitate. We propose a serious game concept that combines such modeling methods within a single intelligent simulation platform so that it becomes a low-threshold interface for continuous knowledge exchange and gain between teachers, students and agents.

1 Introduction

After decades of educational games, today we are witnessing the third generation educational use of computer games: *serious games* [1]. At the 2008 Game Developers Conference, serious game proponents Sawyer and Peters presented a comprehensive taxonomy that shows how serious games are increasingly being embraced for diverse purposes, ranging from training, education and research to advertisement, production and design [2]. The authors show that this trend appears not just in public sectors like government and NGO's, defense, health care and education, but also private sectors for marketing & communications, corporate and industrial applications. In fact, these collaborative human-in-the-loop simulation games, designed either to educate players about particular phenomena or to study human behavior within simulated environments, have become an important medium for many applications and the body of related research is growing rapidly.

[⋆] This research is funded in part by the Next Generation Infrastructures Foundation and in part by the Delft University of Technology.

We propose a new type of serious game in which users explore and delegate their strategies to adaptive agents with intelligent support at their own expertise level. Inspired by agent modeling games like *StarLogo TNG*[1] [3] and the *Neuro-Evolving Robotic Operatives*[2] or NERO game [4], our interest is in combining various agent modeling methods with intelligent tutoring in ill-defined domains such as operations management in infrastructures and marketplaces. By extending the educational *Global Supply Chain Game*[3] or GSCG [5] with a range of intelligent agent and tutoring techniques, we may be able to determine how agent and tutoring technologies can enhance simulation games to help students learn more about strategy in ill-defined problem domains compared to manual operation. In this chapter we argue that the required technologies for intelligent agent modeling games exist, holding great potential for many educational, scientific and societal applications including ill-defined domains.

Educational Settings. Users can design and exchange agents able to make many more decisions than the players could themselves. Since the game simply executes the user-defined agent models it does not need to wait for user input on each decision, the game environment complexity can increase much further as players gain experience. Also, since the behavior is already formalized in a model, the game environment can evaluate the behavior much more effectively to determine whether learning goals are being achieved and provide suitable challenges or hints as is common in intelligent tutoring systems.

Scientific Settings. User-defined agent models provide much richer game data to study than traditional game action histories do, since agent models also provide the reasoning behind the actions. Instead, one may now much more easily track changes in the agent models and observe whether and when learning occurs. Moreover, the formalized models enable the use of statistical methods to generalize behavioral patterns from the user-defined models gathered over many sessions to serve as human representative agents in less expensive simulations that no longer require humans present.

Societal Settings. An easy-to-use behavior modeling language might open up the realm of multi-agent simulation besides computer scientists and students to a large audience of lay users. Non-programmer employees of corporations and governments could still compare and experiment with new strategies in their particular domain, since the modeling language allows them to understand how the agents behave. Similarly, individual consumers would be able to define exactly how they would like their negotiator agent to represent them in for instance electronic markets or legal courts.

To show how implementations of such agent modeling games might be achieved in ill-structured problem domains, we first review some of the existing theory related to intelligent agent modeling in serious games and simulations. We then describe the basic ideas and novelty of our game concept in more detail. Finally we conclude with a summary and discuss future work.

[1] http://education.mit.edu/starlogo-tng/
[2] http://www.nerogame.org/
[3] http://www.gscg.org/

2 Related Theory

We are interested in using intelligent agents both as pedagogical and as analytical tools for education and inquiry purposes by novice and expert users. This section reviews various candidate agent technologies and identifies a number of challenges that arise if we want to integrate them into an intelligent agent modeling game in ill-structured domains such as operations management.

2.1 Learning by Building Agents

Agent modeling has been applied as an educational tool in a number of ways. Most notable is the work by Seymour Papert and colleagues on *Turtle Geometry*, a mathematical programming environment or *microworld* in which kids try to solve geometrical problems by programming a turtle-shaped agent using the LISP-based LOGO[4] language [6]. Many more of these learning environments have emerged since then such as *Boxer*[5] [7], *Alice*[6] [8], *ToonTalk*[7] [9], *Stagecast Creator*[8] [10], *Scratch*[9] [11], *Etoys*[10] [12] and the recent *Kodu Game Lab*[11] [13]. Educators use these environments in elementary and high schools for *constructionist* learning, that is, kids 'construct' their knowledge as they create increasingly complex agents to solve ever more complex tasks. Kids not only learn new ways of tackling problems, that mistakes are essential for debugging your thinking, or that there are often many solutions, but they also gain a new, more personal perspective on the subject matter [6].

Unfortunately, education scholars criticize this approach for its lack of student guidance [14]. Furthermore, while agents are typically used as pedagogical tools in highly structured, well-defined domains such as science, technology, engineering and mathematics, for ill-structured problem domains such as operations management, agent modeling and intelligent tutoring seem problematic. For instance in (serious) games, the modeling effort consists of no more than switching city governor priorities in *Civilization* [15], programming train schedules in *Transport Tycoon* [16], queuing factory production and setting way points in real-time strategy games such as *Dark Reign* [17] or *Homeworld: Cataclysm* [18], setting stock reorder levels in *Industry Masters*[12] [19], etc. We are interested in creating microworlds with higher level strategic agent programming to educate students about the complexity in infrastructures and markets. Typically, only expert (game) agent designers can typically model such complex behaviors to meet today's demand for intelligent and realistic human-like opponents.

[4] http://el.media.mit.edu/logo-foundation/
[5] http://www.soe.berkeley.edu/boxer/
[6] http://www.alice.org/
[7] http://www.toontalk.com/
[8] http://www.stagecast.com/
[9] http://scratch.mit.edu/
[10] http://www.squeakland.org/
[11] http://research.microsoft.com/en-us/projects/kodu/
[12] http://www.industrymasters.com/

Novel modeling environments are starting to appear such as Pogamut[13] [20] or *PMFserv*[14] [21] for beginner or non-computer experts to model their own agents for existing game environments. These however still require some programming expertise and fine tuning from the developer in order to arrive at the desired agent behavior, and the modeling itself has no educational or empirical purpose. Given the limited success of agent modeling in ill-structured problem domains together with the proven success of agent modeling in well-defined domains such as mathematics, physics and games, we propose the following:

Proposal 1. *Provide students with malleable agents that enhance learning in ill-structured problem domains.*

2.2 Making Complex Models Easy

From the social studies such as economy and biology came computational models to describe whole societies and the effects of their interactions on particular global measures such as the distribution of wealth or health. As computation becomes cheaper and more powerful, simulations are moving from differential equations assuming homogeneous populations towards heterogeneous models with much more complexity. According to Parunak and colleagues, "*agent-based modelling is most appropriate for domains characterised by a high degree of localisation and distribution and dominated by discrete decision*" [22].

Agent-based modeling is an approach that has gained great popularity in social and economic sciences to study and educate about phenomena emerging from the interactions of large heterogeneous populations. The recent establishment of the Open Agent Based Modeling Consortium[15] [23] mirrors the field's growing popularity. Well-known examples of such agent-based models (ABMs) include John Conway's *Game of Life*, Wolfram's class of cellular automata, Epstein and Axtell's *SugarScape* [24], and today topics range from the evolution of epidemics, traffic jams to the self-organization of ants, sensor networks and unmanned aerial vehicles.

Papert's microworlds evolved into increasingly powerful agent-based modeling and simulation engines. Nikolai and Madey survey many more agent based modeling toolkits [25] such as *StarLogo* [26], *NetLogo*[16] [27], MASON[17] [28] and *RePast*[18] [29] which are widely used in social and life sciences. Their successful application in empirical work is also due to the user-friendly interfaces that make programming and debugging agents relatively easy. Thanks to the transparency of the agent models and of their effect on emergent social properties through their interactions, we are gaining insight on how to design and control robust collectives of simple individuals that together act intelligently.

[13] http://artemis.ms.mff.cuni.cz/pogamut/
[14] http://www.seas.upenn.edu/ barryg/HBMR.html
[15] http://www.openabm.org/
[16] http://ccl.northwestern.edu/netlogo/
[17] http://cs.gmu.edu/~eclab/projects/mason/
[18] http://repast.sourceforge.net/

Some of the agent modeling environments mentioned earlier, including *StarLogo TNG* [3], *Alice* [8], *ToonTalk* [9], *Stagecast Creator* [10], *Scratch* [11], *Etoys* [12] and *Kodu Game Lab* [13], provide a visual programming language simple enough for children to create their own games and animations. Most visual languages are based on the LogoBlocks research developed at MIT, which later evolved into OpenBlocks [30]. These are powerful languages that make efficient use of shapes and colors to represent command structures, procedures and data types. However, the programming blocks usually map directly onto traditional programming commands. Interesting addition to such languages might be multiple complexity layers in the block semantics and control over learning mechanisms that agents may use to learn and improve their own behavior.

Proposal 2. *Provide students with clear and simple interfaces that show how their agent models contribute to overall system performance.*

2.3 Validating Interactive Simulations

Simulation games are also applied increasingly as *research method*, either to study or even to capture the richness of human behavior in complex systems through human-in-the-loop simulation. By examining how players (subjects) behave, researchers can potentially replicate their behavior in agent-based simulations that would otherwise contain only simple heuristic rules and assumptions. For instance, the *Trust and Tracing* game is used to study cultural differences in trade by playing the game with players of different backgrounds and the *Mango Chain Game* studies the searching and bargaining sources of transaction costs [31]. The next step could be to use such games as interfaces to elicit or capture expert knowledge on how to behave in complex multi-actor systems.

Unfortunately, the simulation engines mentioned above which are frequently used by sociologists, biologists and economists have little or no specification of the simulation formalisms that describe the underlying assumptions. Existing formalisms could clearly separate models from their reference or source systems as well as from the simulator as Zeigler and colleagues propose in their theory of modeling and simulation [32]. In fact, the authors argue that the discrete event system specification (DEVS) formalism is the most general type which can be applied to cellular automata simulations as implemented in aforementioned simulation engines (which is possible if one considers the cells to be agents). For example, Müller recently presented a formal semantics for DEVS-based multi-agent simulation [33].

Dolk proposes to "*apply model management design principles to design a* generalized agent-based modeling environment *(GAME) with ABMS language(s) and accompanying interfaces that support the entire spectrum of potential users: programmer, analyst, decision-maker, and also cross-fertilizes with existing OR/MS modeling paradigms*" [34]. Similarly, Schut proposes a generalized agent-based simulation approach [35]. By adopting a single simulation formalism that separates a model from its simulator, the following issues could be resolved:

– *reusability*, the simulation engine can be applied to new models and vice versa (for instance run NetLogo models in Repast);
– *maintenance*, the engine and/or model can be maintained separately;
– *consistency*, the models can be proven to have certain properties before running any simulation; and
– *validity*, the model can be validated and verified on other simulators.

Creating a programming environment for agent behaviors that suits any type of domain will require a generalized behavior modeling and simulation formalism. Dolk also mentions a caveat: "*Model management was successful in capturing structural dimension of models,* not *the dynamic aspects prevalent in simulation*" [34, p.6]. Furthermore, the formalism must accommodate procedural commands (if-then, for-next, etc.), domain specific concepts (strategy-x, object-y, etc.), as well as model adaptation due to learning or evolution, so a combination of formalisms may perhaps be a better solution. In order to have generalizable results from our simulation game concept, we propose the following:

Proposal 3. *Adopt general simulation formalisms to ensure validity of models across simulation environments and studies.*

2.4 Providing High-Level Reasoning

From the fields of Distributed Artificial Intelligence and Cognitive Psychology many frameworks have emerged that support the design and validation of complex organizations with (multiple) intelligent agents to respectively mimic or study their (collective) intelligence. Wooldridge and Jennings define these agents as processes implemented on a computer that have *autonomy* (they control their own actions); *social ability* (they interact with other agents through some kind of 'language'); *reactivity* (they can perceive their environment and respond to it); and *pro-activity* (they are able to undertake goal-directed actions) [36].

The Multi-Agent Systems (MAS) paradigm is already being adopted as a new framework for software development to extend the service-oriented approach for developing robust scalable software systems. Thanks to agent middleware or platforms such as JADE[19] [37] or *AgentScape*[20] [38], autonomous and mobile agents can go out on the Internet to interact on their owner's behalf. Today entire organizations are modeled and validated thanks to human-readable reasoning frameworks. A widely used approach is the *Beliefs-Desires-Intentions* or BDI paradigm applied in frameworks like DESIRE [39], *AgentSpeak* [40] (implemented in the JACK[21] [41] and JASON[22] [42] platforms), GOAL[23] [43] and 2APL[24] [44].

[19] http://sharon.cselt.it/projects/jade/
[20] http://www.agentscape.org/
[21] http://www.agent-software.com.au/products/jack
[22] http://jason.sourceforge.net/
[23] http://mmi.tudelft.nl/~koen/goal.html
[24] http://www.cs.uu.nl/2apl/

These frameworks offer advanced reasoning methods for agents and help to produce human-readable explanations of agent actions useful in human-agent collaboration. For example, air force pilots could delegate reconnaissance tasks to a group of BDI agents (e.g. unmanned aerial vehicles) and keep updated on their goals and plans during a mission. This behavioral transparency makes the delegate agents dependable parts of a cooperative man-machine team during critical situations. Such *explainable artificial intelligence* (XAI) has already proven useful in human-robot collaboration as well as for education purposes, for instance in tutoring tactical behavior [45] or virtual firemen training [46].

However, due to their complexity, their computational overhead and their limited or lack of visual programming interface, programming agents with these languages is currently suited only for computer programmers and scientists rather than our intended players of (serious) games. Using a different modeling approach, an interesting project that avoids these problems is *Betty's Brain*[25] developed by [47] and [48] where students design concept models to represent their current knowledge and teach an agent to solve reasoning tasks in role-playing games. These concept models serve as scaffolds to help students structure their own knowledge and gain a deeper understanding. Leelawong and Biswas report that this approach taught students meta-cognitive skills which students successfully transferred to new learning environments [49]. The models can also be analyzed statistically to study student learning behaviors [50]. Given these results we are convinced that automated reasoning techniques for human-agent collaboration are viable education methods in games. We propose to:

Proposal 4. *Employ standardized reasoning methods readable by both machines and humans to increase generalizability.*

2.5 Learning with Intention

While human-readable agent reasoning has great value across application domains like human-robot collaboration and education, the popular BDI framework has two main drawbacks: a lack of learning competencies and of explicit multi-agent functionality. Guerra-Hernandez and colleagues try to overcome these deficiencies by letting agents adapt the reasoning process using logical decision trees [51]. Lokuge and Alahakoon also extend BDI with learning capability for container terminal planning but use an adaptive neuro-fuzzy inference system which applies neural networks to optimize fuzzy agent models [52].

Besides regression with trees or neural nets, another approach to adaptive BDI may be to apply population-based *evolutionary* algorithms. These apply variation and selection mechanisms, principles of evolution known to be successful for finding innovative solutions in a large variety of problems ranging from design and art to scheduling and control. Especially in complex, poorly understood, nonlinear, multi-objective domains, the basic mechanisms of evolution have proven to be powerful optimization tools [53].

[25] http://www.teachableagents.org/

There are different dialects in evolutionary algorithms, including genetic algorithms, evolutionary strategies and symbolic regression. Genetic algorithms in particular have been combined with other machine learning paradigms to solve (agent) control problems. Genetic algorithms can be integrated with neural networks to produce *evolving neural networks* or *neuroevolution*, combining the innovative power of evolution with the pattern recognition power of the neural nets [54]. These evolved neural nets typically optimize agent control, for instance in collective behavior tasks like construction [55]. Another example is the NERO game, in which players evolve teams of specialist soldiers to compete against other teams [4]. Neural networks are hardly readable even for experts, let alone for novices and non-computer scientists. Therefore this evolutionary method, although proven successful, appears unsuitable for educational and inquiry purposes where transparent agent models are desirable.

Genetic algorithms can be combined with production rules to become *learning classifier systems* which produce control models that are much more transparent due to the readable if-then structure of the evolved rule sets [56]. Evolved rule sets may be less accurate than evolved neural nets, but in return provide clearer insight into how and why the controller works or not. We suggest that in a similar fashion, adaptive BDI controllers can be made using evolutionary optimization while respecting the BDI paradigm. Although this hypothesis requires further study, we propose the following:

Proposal 5. *Enable the agent system to self-improve using machine learning algorithms for behavior optimization.*

3 The Agent Modeling Game Concept

This section introduces our agent modeling game concept that integrates several of the agent and learning technologies outlined above. The system is an intelligent simulation game that tutors students as they model their delegate agents directly, using visual programming, or indirectly, using machine learning techniques such as evolution or imitation. We also discuss the system's components.

3.1 Integrating Direct and Indirect Agent Modeling

We have seen a number of techniques for modeling agents: *direct* modeling techniques where students use visual programming languages to edit the agent model, but also *indirect* modeling methods where students simply reward observed behavior as the agent models evolve, or students enact the desired behavior from which an agent model is generalized or imitated.

Direct Agent Modeling. Visual agent modeling languages are gaining momentum thanks to environments like *StarLogo TNG* [3], *Alice* [8] and *Kodu Game Lab* [13]. Their success is likely due to the clarification of typical coding conventions with familiar metaphors. For instance, colors and shapes represent types of variable and object, agent states include 'happy' and 'sad', etc.

Still, we noted above some challenges that we are interested in resolving. The first is a typical lack of general systems specifications, which in case of 'realistic' simulations could facilitate external validation of the simulation results through reproduction. We suggested the DEVS formalism as a probable candidate. The second limitation is the lack of more abstract, higher-level reasoning in these environments such as the Beliefs-Desires-Intentions paradigm. Thirdly, we mentioned the limitation of these environments to well-defined domains such as maths, physics and programming education. Another important addition to agent programming microworlds might be intelligent tutoring to provide students with more guidance as they explore the simulated environment. And finally, current (visual) agent programming languages feature little autonomous knowledge discovery by the agents, even though there seem to be many suitable machine learning algorithms readily applicable.

For domain experts or teachers, building a comprehensive agent model directly as described above may be a suitable method to build an agent that performs particular by-the-book or typical strategies for students to play with or compete against. Except for the intelligent tutor providing much needed student guidance as they model their agent, this mode of agent modeling is fairly standard. Nonetheless, for some this is arguably a rather challenging approach to creating an intelligent delegate agent, especially for students who may have little or no experience with programming nor with the domain that the agent must tackle. We are thus also interested in developing possibilities where one explores a domain with delegate agents without directly dealing with the agent model.

Indirect Agent Modeling Challenges. What makes our intelligent tutoring game concept different from current genres (serious) games is the extension of (visual) agent programming with evolutionary agent modeling techniques, inspired in particular by the Neuro-Evolving Robotic Operatives[26] or NERO game by Stanley and colleagues [4]. In their novel kind of game, the user's objective is to train an army by designing increasingly complex learning environments in a *sand box*. Here intelligent agents evolve to optimize a user-defined reward function, before being combined into an army containing various specialties that will battle another army in a more complex environment. Such coaching by designing increasingly complex challenges is known in biology as *shaping* [57]. In stead of telling each soldier exactly what to do or programming their behavior to the letter, users become drill sergeants and think about how best to train their agents so they can be trusted to execute the desired strategy. We will investigate applying this method to the serious games.

Another approach to creating visibly intelligent agent behavior is training the agents to imitate human behavior as is done in academic strategy games [58] or even commercial first-person shooters [59]. Unfortunately these advanced imitation approaches typically train *off-line*, that is between actual game sessions, and may be less suited for interactively training agents.

[26] http://www.nerogame.org/

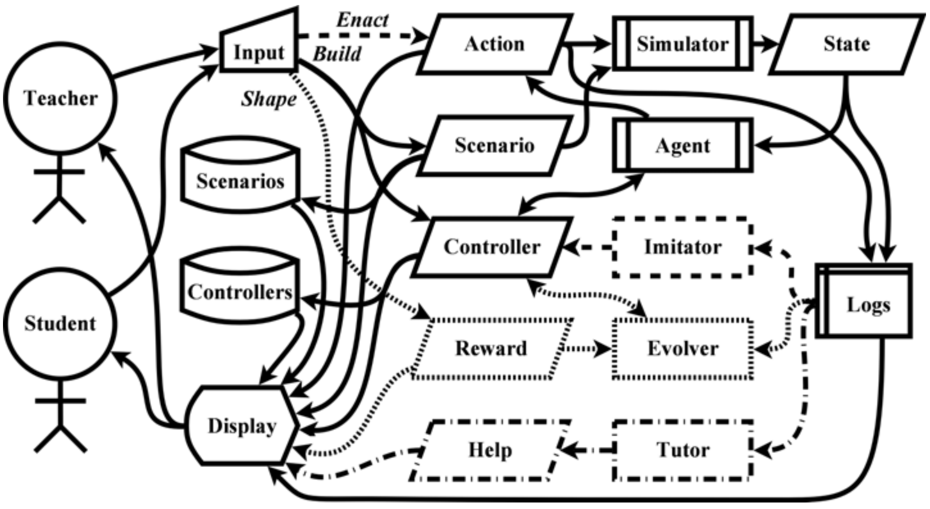


Fig. 1. Intelligent simulation game concept where students *build* controllers for their delegate agents, *enact* the behavior to be imitated (dashes) or *shape* an agent controller as it is evolved (dots) with help from a tutor (dot-dashes)

By contrast, educational environments also exist where students interactively train an agent *on-line*. For instance in *Sophie's Kitchen* [60] students teach a learning agent by example how to bake a cake. Some interesting lessons were learned in this project, for instance that people use reward channels not only for feedback, as is common in machine learning, but also for future-directed guidance and motivation [61]. Whether this result holds when training advanced agents such as those of the NERO game remains an open research question.

3.2 System Components

Our intelligent agent modeling game concept as shown in Fig. 1 includes users (circles), fixed and dynamic processes (rectangles), data objects (parallelograms), internal store (squares) and external stores (cylinders), a display and a manual input. First we explain the data objects and stores:

Action. In simulation games, usually teachers or students play the role of particular actors and select actions to perform. In our concept agents can take over and perform actions according to a preprogrammed strategy represented within its controller. Actions are stored internally in the **Logs** store.

Scenario. Teachers can build test scenarios containing environments or worlds with learning goals that students must meet using their agents. Students can also build training scenarios with incrementally complex worlds in which to evolve their agents. Scenarios are stored externally in the **Scenarios** store.

Controller. The core objects of this system are agent controllers. They describe how agents should act in various world states. Teachers can build controllers for agents serving as adversaries or to provide exemplar behavior for students. Students can build controllers to cope with various adversaries, roles and scenarios. Controllers can also be created through imitation or evolution. Controllers are stored externally in the **Controllers** store.

Reward. Students can reward an agent's controller based on observed performance in order to evolve its behavior. Controllers receiving higher rewards get a better chance of survival during the game and thus for recombination to produce offspring (with some mutations) that later might replace them.

State. The world state describes the current condition of the simulated environment and all agents within it. The simulator determines the next world state based on its current state, the world model and the selected agent actions. To provide a historic perspective, to improve tutor guidance, to imitate stored behavior and to evolve improved controllers, world states are stored internally in the **Logs** store.

Help. Tutoring guidance or help contains hints for the student to solve current challenges. This advise may be derived from agent behavior logs stored previously. Help is provided as students enact the desired behavior or as they build new controllers.

Components present in any agent-based simulator are the simulator, agent, input and display. Processes that make this simulation game concept *intelligent* are the tutor, the imitator and the evolver.

Simulator. The system's core process is the simulator which takes a world model and actions selected by agents or students to generate future world states. It must be robust and suited to host many different types of simulation environments such as cellular automata, graphs or 3D spaces. We are planning to implement the general DEVS formalism using the DSOL[27] (Distributed Simulation Object Library) framework [62].

Agent. Agents observe the world state and require a controller to derive the next action to perform. The bidirectional arrow implies that (adaptive) agents may modify their controller during a simulation.

Input and Display. Based on current and previous world states, the input and display provide an interactive representation of the game status. When the teacher or student is not selecting new worlds or agent controllers, this interface displays the world state, agent controller and performed action, performance statistics, the current reward settings used to shape agent controller evolution, student learning progress in the current world, and for students possibly some tutor help. Rule activation may be visualized such that users are able to determine what part of the agent's controller is responsible for the delegate agent's successes or failures.

Tutor. Intelligent tutors typically contain a teacher model, a student model, and a domain model. This component derives a student model from the student behavior stored in the logs and provides appropriate remedial feedback such as hints and suggestions. This help is based on a teacher model that is

[27] http://www.simulation.tudelft.nl/

generalized from stored behaviors of other students. This way the tutor can self-improve its help as more students play the current world. Given that the scenarios will easily become very complex, the difficulty will lie in determining the user's current skill level and which existing agent controllers have tackled relevant problems which the user is currently unable to solve.

Evolver. The evolver recombines and mutates agent controllers, selecting new agents that optimize some user-defined reward or fitness function. Naturally, fitness functions will vary across domains and designing this function is critical to defining the limits of the evolutionary approach. The user also provides training scenarios with incremental difficulty levels that to challenge their agents such that they learn the desired behavior. The interface should provide an easy-to-use scenario editor for this process to add value as alternative agent modeling method.

Imitator. This process induces an agent controller that imitates some desired behavior. Generalizing student decisions to new, unseen situations requires some example behavior enacted by the user. From a machine learning perspective, this may be one of the most challenging processes to implement. However this approach is valuable particularly for inquiry with automatically generated models of actual human behavior.

More detailed information as to each component's operation are purposely left out of this general concept and can be filled in by game designers in whatever way they see fit.

4 Conclusion

This chapter presented a new serious game concept that revolves around tutored end-user modeling of intelligent agents for knowledge transfer (education) and discovery (inquiry). Simulation platforms that support end-user modeling of intelligent agents may not only prove to be powerful inquiry methods but also methods especially suited in education, for as the Latin proverb says: *docendo discimus* (by teaching we learn).

To realize such a concept we made a number of proposals that are based on combining existing solutions and frameworks from various research areas. We propose the adoption of a single simulation formalism such as DEVS, the adoption of a human-readable reasoning method such as BDI, the integration of human-readable reasoning methods with evolutionary algorithms such as classifier systems, and finally the adoption of a visual programming language generated from a shared ontology.

There remain some important questions that need answering before we know whether this is a successful approach for gaining and transferring knowledge. For instance: can users actually apply building, shaping and imitation, do they want to, and what kind of results can we expect in terms of validity and performance? In future work we hope to answer these questions.

References

1. Egenfeldt-Nielsen, S.: Third generation educational use of computer games. Journal of Educational Multimedia and Hypermedia 16(3), 263–281 (2007)
2. Sawyer, B., Smith, P.: Serious games taxonomy. In: GDC 2008: Proc. Game Developers Conf., San Francisco, CA, USA, February 19 (2008)
3. Wang, K., McCaffrey, C., Wendel, D., Klopfer, E.: 3D game design with programming blocks in StarLogo TNG. In: ICLS 2006: Proc. 7th Int'l Conf. on Learning sciences, International Society of the Learning Sciences, pp. 1008–1009 (2006)
4. Stanley, K.O., Bryant, B.D., Miikkulainen, R.P.: Real-time neuro-evolution in the NERO video game. IEEE Trans. Evolutionary Computation 9(6), 653–668 (2005)
5. van Houten, S.P.A.: A suite for developing and using business games: supporting supply chain business games in a distributed context. PhD thesis, Delft University of Technology, Delft, The Netherlands (November 6, 2007)
6. Papert, S.A.: Mindstorms: Children, Computers, and Powerful Ideas. Basic Books, Inc., New York (1980)
7. di Sessa, A.A., Abelson, H.: Boxer: a reconstructible computational medium. Communications of the ACM 29(9), 859–868 (1986)
8. Pausch, R.: Alice: Rapid prototyping for virtual reality. IEEE Computer Graphics and Applications 15(3), 8–11 (1995)
9. Kahn, K.: Toontalk^{TM}—an animated programming environment for children. Journal of Visual Languages & Computing 7(2), 197–217 (1996)
10. Smith, D.C., Cypher, A., Tesler, L.: Programming by example: novice programming comes of age. Communications of the ACM 43(3), 75–81 (2000)
11. Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B., Resnick, M.: Scratch: a sneak preview. In: C5 2004: Proc. 2nd Int'l Conf. on Creating, Connecting and Collaborating through Computing (2004)
12. Kay, J.: Learner control. User Modeling and User-Adapted Interaction 11(1-2), 111–127 (2004)
13. Lerner, E.: Kodu doesn't have realistic graphics, huge explosions, or even a way to win. But it just might change the way we think about the world. Seed Magazine (June 23, 2009)
14. Kirschner, P.A., Sweller, J., Clark, R.E.: Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. Educational Psychologist 41(2), 75–86 (2006)
15. Meier, S.: Sid Meier's Civilization. MicroProse (1991), `http://www.civilization.com/`
16. Sawyer, C.: Transport Tycoon. MicroProse (1994), `http://www.chrissawyergames.com/`
17. Auran: Dark Reign. Activision (September 1997), `http://www.auran.com/games/darkreign/`
18. Relic Entertainment: Homeworld: Cataclysm. Sierra Studios (June 2000)
19. Lehnert, T., Maiburg, M., Rimmer, B.: Industry masters. Tycoon Systems (2007)
20. Gemrot, J., Kadlec, R., Bída, M., Burkert, O., Píbil, R., Havlíçek, J., Şimloviç, J., Vansa, R., Ştolba, M., Zemçák, L., Brom, C.: Pogamut 3 can assist developers in building AI for their videogame agents. In: [63]
21. Silverman, B., Chandrasekaran, D., Weyer, N., Pietrocola, D., Might, R., Weaver, R.: NonKin Village: A training game for learning cultural terrain and sustainable counter-insurgent operations. In: [63]

22. Van Dyke Parunak, H., Savit, R., Riolo, R.L.: Agent-based modeling vs. Equation-based modeling: A case study and users' guide. In: Sichman, J.S., Conte, R., Gilbert, N. (eds.) MABS 1998. LNCS (LNAI), vol. 1534, pp. 10–25. Springer, Heidelberg (1998)
23. Janssen, M.A., Alessa, L.N., Barton, M., Bergin, S., Lee, A.: Towards a community framework for agent-based modelling. Journal of Artificial Societies and Social Simulation 11(2), 6 (2008)
24. Epstein, J., Axtell, R.: Growing Artificial Societies: social science from the bottom up. MIT Press, Cambridge (1996)
25. Nikolai, C., Madey, G.: Tools of the trade: A survey of various agent based modeling platforms. Journal of Artificial Societies and Social Simulation 12(2) (2009)
26. Resnick, M.: Decentralized Modeling and Decentralized Thinking. In: Modeling and Simulation in Precollege Science and Mathematics, pp. 114–137. Springer, New York (1999)
27. Wilensky, U.: Modeling nature's emergent patterns with multi-agent languages. In: Futschek, G. (ed.) EuroLogo 2001: A turtle odyssey: Proc. 8th Eur. Logo Conf., Linz, Austria, Österreichische Computer Gesellschaft, August 21-25 (2001)
28. Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., Balan, G.: MASON: A multi-agent simulation environment. Simulation 81(7), 517–527 (2005), `http://www.cs.gmu.edu/~eclab/projects/mason/`
29. North, M.J., Collier, N.T., Vos, J.R.: Experiences creating three implementations of the repast agent modeling toolkit. ACM Trans. Modeling and Computer Simulation 16(1), 1–25 (2006)
30. Roque, R.V.: OpenBlocks: An extendable framework for graphical block programming systems. Master's thesis, MIT Dept. of Electrical Engineering and Computer Science, Cambridge, MA, USA (May 2007)
31. Meijer, S.A.: The organisation of transactions – Studying supply networks using gaming simulation. PhD thesis, Wageningen Universiteit, Wagningen, The Netherlands (March 4, 2009)
32. Zeigler, B.P., Praehofer, H., Kim, T.G.: Theory of Modeling and Simulation—Integrating Discrete Event and Continuous Complex Dynamic Systems, 2nd edn. Academic Press, Inc., San Diego (2000)
33. Müller, J.P.: Towards a formal semantics of event-based multi-agent simulations. In: MABS 2008: Proc. 9th Int'l AAMAS, Workshop on Multi-Agent-Based Simulation, Estoril, Portugal, May 12-13 (2008)
34. Dolk, D.R.: Model management for agent-based modelling and simulation. In: IFIP WG 7.6 Workshop on Modelling and decision support for network-based services, Warsaw, Poland, September 1-3 (2008)
35. Schut, M.C.: On model design for simulation of collective intelligence. Information Sciences (in press, 2009)
36. Wooldridge, M., Jennings, N.R.: Intelligent agents: Theory and practice. Knowledge Engineering Review 10(2), 115–152 (1995)
37. Bellifemine, F., Poggi, A., Rimassa, G.: Developing multi-agent systems with JADE. In: Castelfranchi, C., Lespérance, Y. (eds.) ATAL 2000. LNCS (LNAI), vol. 1986, p. 89. Springer, Heidelberg (2001)
38. Brazier, F.M.T., Mobach, D.G.A., Overeinder, B.J., van Splunter, S., van Steen, M., Wijngaards, N.J.E.: AgentScape: Middleware, resource management, and services. In: SANE 2002: Proc. 3rd Int'l Conf. on System Administration and Network Engineering, Maastricht, The Netherlands, May 27-31, pp. 403–404 (2002)

39. Brazier, F., Dunin-Keplicz, B., Jennings, N.R., Treur, J.: Formal specification of multi-agent systems: A real-world case. In: Lesser, V., Gasser, L. (eds.) Proc. 1st Int'l on MultiAgent Systems, San Francisco, California, USA, June 12-14, pp. 25–32. MIT Press, Cambridge (1995)
40. Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In: Perram, J., Van de Velde, W. (eds.) MAAMAW 1996. LNCS, vol. 1038, pp. 42–55. Springer, Heidelberg (1996)
41. Busetta, P., Rönnquist, R., Hodgson, A., Lucas, A.: Light-weight intelligent software agents in simulation. In: SimTecT 1999: Proc. Simulation Technology and Training Conf., Melbourne, Australia, 29 March - 1 April 1999. Simulation Industry Association of Australia (1999)
42. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak using Jason. John Wiley & Sons, Chichester (2007)
43. Hindriks, K.: Modules as policy-based intentions: Modular agent programming in GOAL. In: Dastani, M.M., El Fallah Seghrouchni, A., Ricci, A., Winikoff, M. (eds.) ProMAS 2007. LNCS (LNAI), vol. 4908, pp. 156–171. Springer, Heidelberg (2008)
44. Dastani, M.: 2APL: a practical agent programming language. Autonomous Agents and Multi-Agent Systems 16(3), 214–248 (2008)
45. van Lent, M., Fisher, W., Mancuso, M.: An explainable artificial intelligence system for small-unit tactical behavior. In: IAAI 2004: Proc. 16th Conf. on Innovative Applications of Artificial Intelligence, San Jose, CA, USA, July 27-29, pp. 900–907. AAAI Press, Menlo Park (2004)
46. Harbers, M., den Bosch, K.V., Meyer, J.J.: A methodology for developing self-explaining agents for virtual training. In: AAMAS 2009: Proc. 8th Int'l Conf. on Autonomous Agents and Multi-Agent Systems, IFAAMAS, Budapest, Hungary, May 10-15, pp. 373–375 (to appear, 2009)
47. Biswas, G., Katzlberger, T., Bransford, J.D., Schwartz, D.L.: TAG-V: Extending intelligent learning environments with teachable agents to enhance learning. In: Moore, J.D., Redfield, C.L., Johnson, W.L. (eds.) Proc. 10th Int'l Conf. on AI in Education: AI-ED in the Wired and Wireless Future, San Antonio, TX, USA, May 19-23, pp. 389–397. IOS Press, Amsterdam (2001)
48. Katzlberger, T.: Learning by Teaching Agents. PhD thesis, Vanderbilt University, Nashville, TN, USA (December 2005)
49. Leelawong, K., Biswas, G.: Designing learning by teaching agents: The Betty's Brain system. Int. J. of Artificial Intelligence in Education 18(3), 181–208 (2008)
50. Jeong, H., Gupta, A., Roscoe, R., Wagster, J., Biswas, G., Schwartz, D.: Using hidden markov models to characterize student behaviors in learning-by-teaching environments. In: Woolf, B.P., Aïmeur, E., Nkambou, R., Lajoie, S. (eds.) ITS 2008. LNCS, vol. 5091, pp. 614–625. Springer, Heidelberg (2008)
51. Guerra-Hernández, A., Ortíz-Hernández, G.: Toward BDI sapient agents: Learning intentionally. In: Mayorga, R.V., Perlovsky, L.I. (eds.) Toward Artificial Sapience: Principles and Methods for Wise Systems, pp. 77–91. Springer, Heidelberg (2007)
52. Lokuge, P., Alahakoon, D.: Improving the adaptability in automated vessel scheduling in container ports using intelligent software agents. Eur. J. of Operational Research 177(3), 1985–2015 (2007)
53. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Springer, Berlin (2003)
54. Miikkulainen, R., Stanley, K.O.: Evolving neural networks. In: [64], pp. 2829–2848
55. van Krevelen, D.W.F., Nitschke, G.S.: Neuro-evolution for a gathering and collective construction task. In: [64], pp. 225–232

56. Lanzi, P.L.: Learning classifier systems: then and now. Evolutionary Intelligence 1(1), 63–82 (2008)
57. Hilgard, E.R., Bower, G.H.: Theories of Learning, 4th edn. Prentice-Hall, Englewood Cliffs (1975)
58. Bryant, B.D.: Evolving visibly intelligent behavior for embedded game agents. PhD thesis, University of Texas, Austin, TX, USA (2006)
59. Priesterjahn, S.: Online imitation and adaptation in modern computer games. PhD thesis, University of Paderborn, Paderborn, Germany (December 2007)
60. Thomaz, A.L.: Socially guided machine learning. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA (June 2006)
61. Thomaz, A.L., Breazeal, C.: Teachable robots: Understanding human teaching behavior to build more effective robot learners. Artificial Intelligence 172(6-7), 716–737 (2008)
62. Jacobs, P.H.M.: The DSOL simulation suite. PhD thesis, Delft University of Technology, Delft, The Netherlands (November 15, 2005)
63. Dignum, F., Bradshaw, J., Silverman, B., van Doesburg, W. (eds.): Agents for Games and Simulations. LNCS (LNAI), vol. 5920. Springer, Heidelberg (2009)
64. Keijzer, M., Antoniol, G., Bates Congdon, C., Deb, K., Doerr, B., Hansen, N., Holmes, J.H., Hornby, G.S., Howard, D., Kennedy, J., Kumar, S., Lobo, F.G., Miller, J.F., Moore, J., Neumann, F., Pelikan, M., Pollack, J., Sastry, K., Stanley, K., Stoica, A., Talbi, E.G., Wegener, I. (eds.): GECCO 2008: Proc. 10th Genetic and Evolutionary Computation Conf., Atlanta, GA, USA, July 11-16. ACM Press, New York (2008)

Author Index